ECE 448
Lecture 3

Combinational-Circuit Building Blocks

Data Flow Modeling of Combinational Logic

Reading

Required

- P. Chu, *FPGA Prototyping by VHDL Examples*

  Chapter 3, RT-level combinational circuit

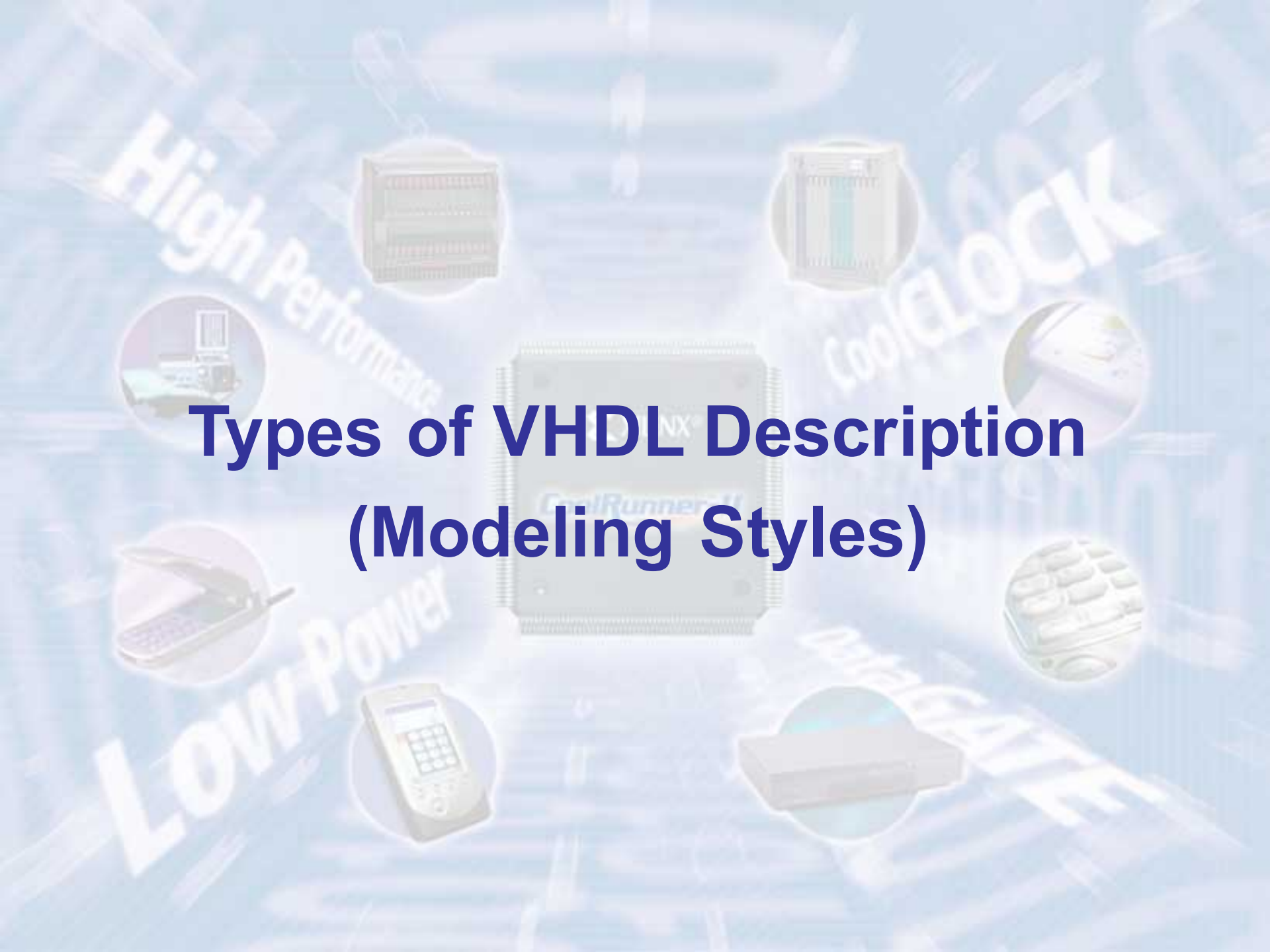
  Sections 3.1, 3.2, 3.3, 3.6, 3.7.1, 3.7.3.

Recommended

- S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design*

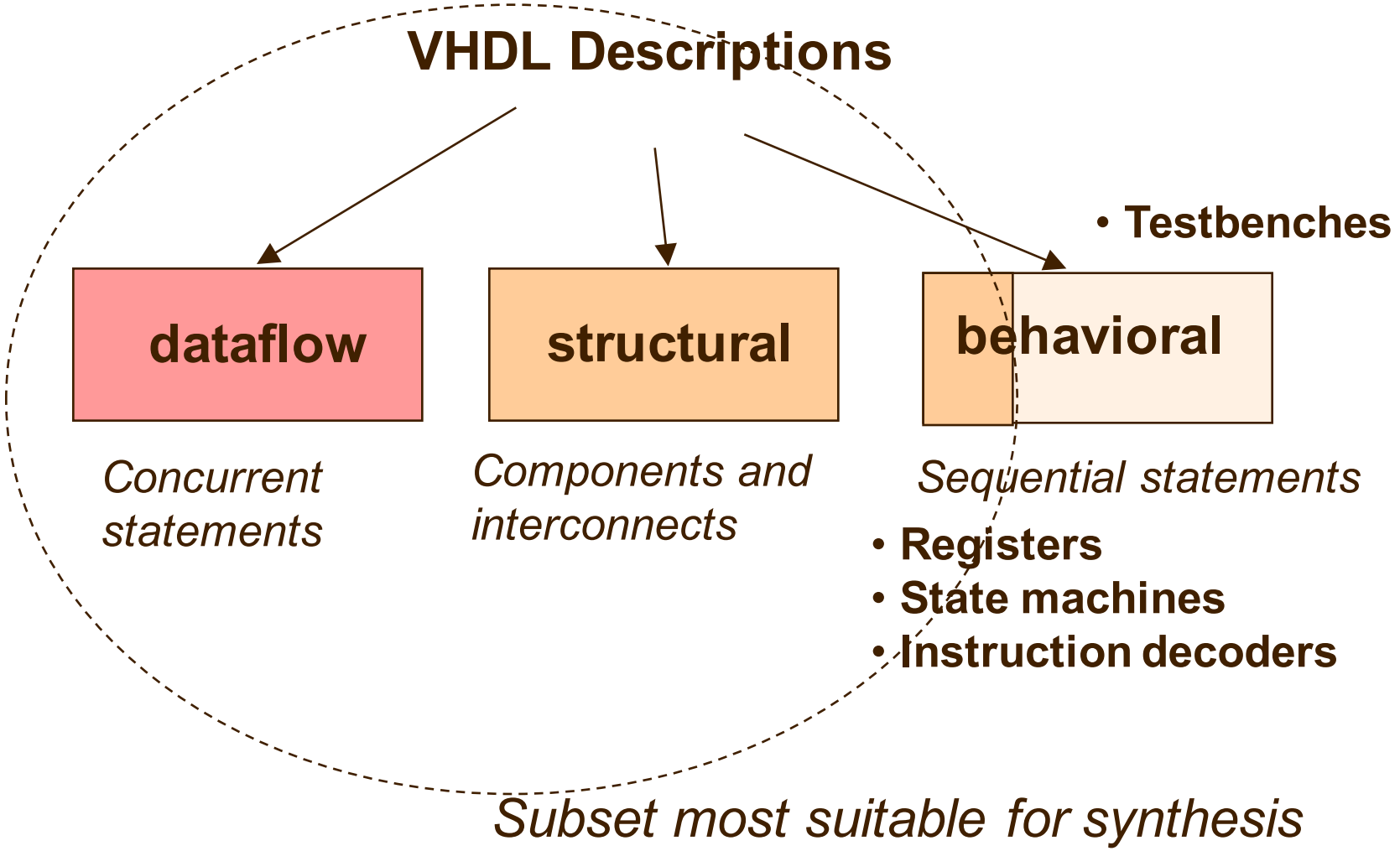
  Chapter 6, Combinational-Circuit Building Blocks

  Chapter 5.5, Design of Arithmetic Circuits Using CAD Tools

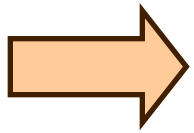


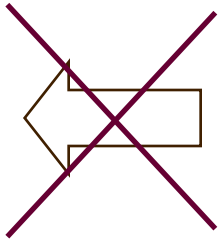
Types of VHDL Description (Modeling Styles)

Types of VHDL Description



Synthesizable VHDL

Dataflow VHDL  VHDL code synthesizable

Dataflow VHDL  VHDL code synthesizable

Data-Flow VHDL

Concurrent Statements

- concurrent signal assignment
  ()
- conditional concurrent signal assignment
  (when-else)
- selected concurrent signal assignment
  (with-select-when)

Concurrent signal assignment

<=

```
target_signal <= expression;
```

Conditional concurrent signal assignment

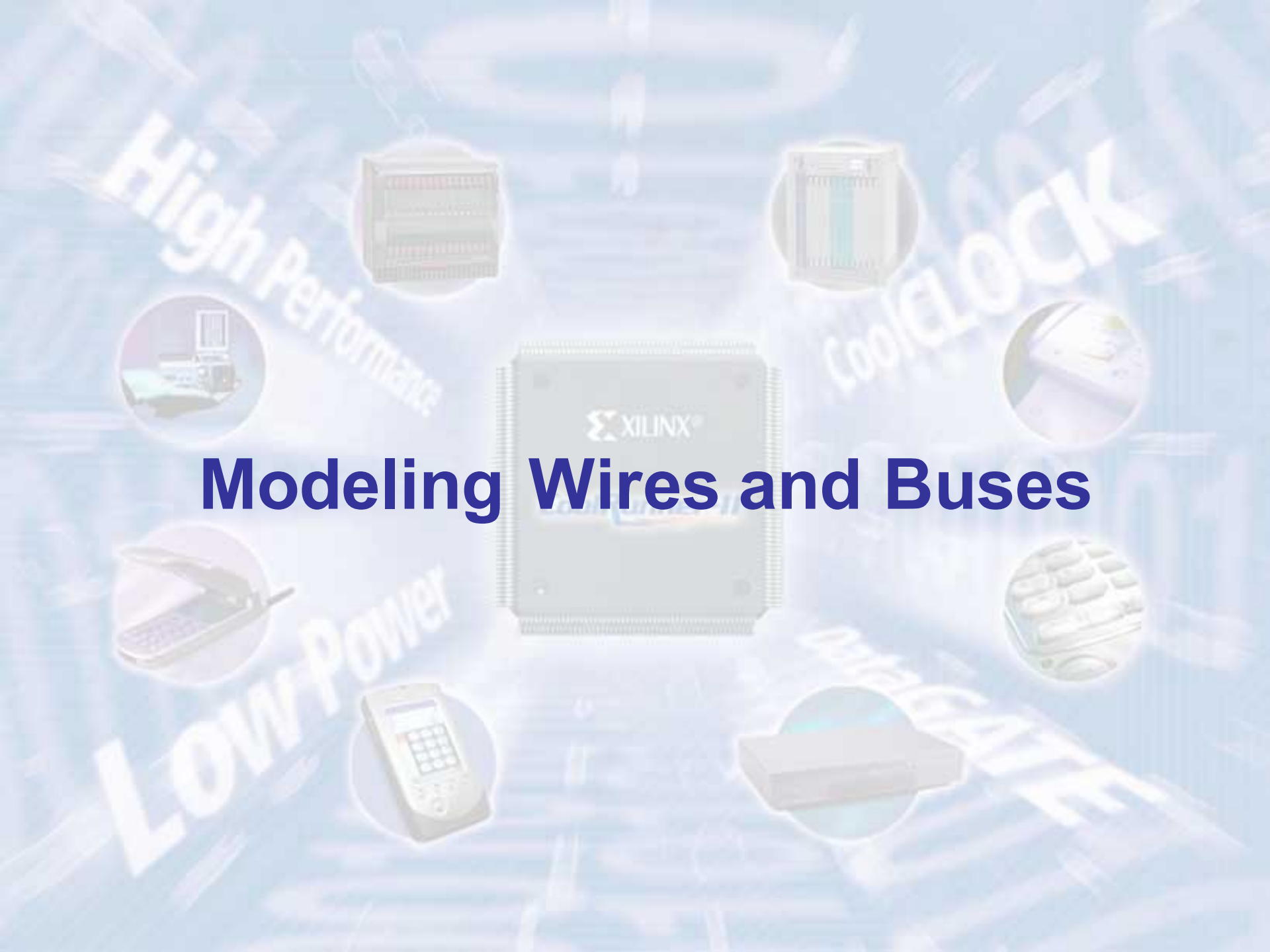
When - Else

```
target_signal <= value1 when condition1 else
                 value2 when condition2 else
                   . . .
                 valueN-1 when conditionN-1 else
                 valueN;
```

Selected concurrent signal assignment

With –Select-When

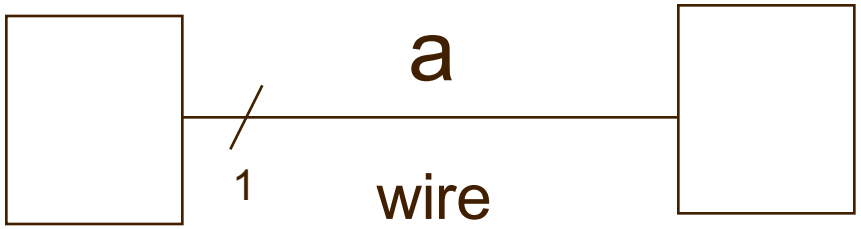
```
with choice_expression select
   target_signal <= expression1 when choices_1,
                    expression2 when choices_2,
                        . . .
                    expressionN when choices_N;
```



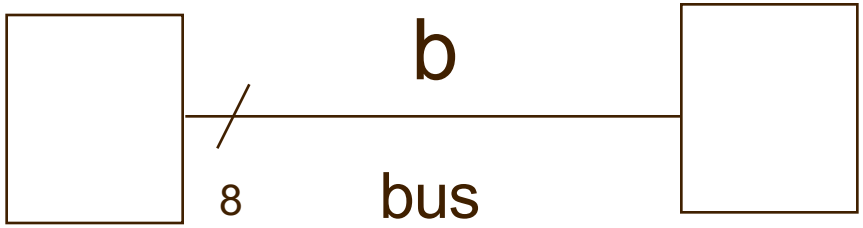
Modeling Wires and Buses

Signals

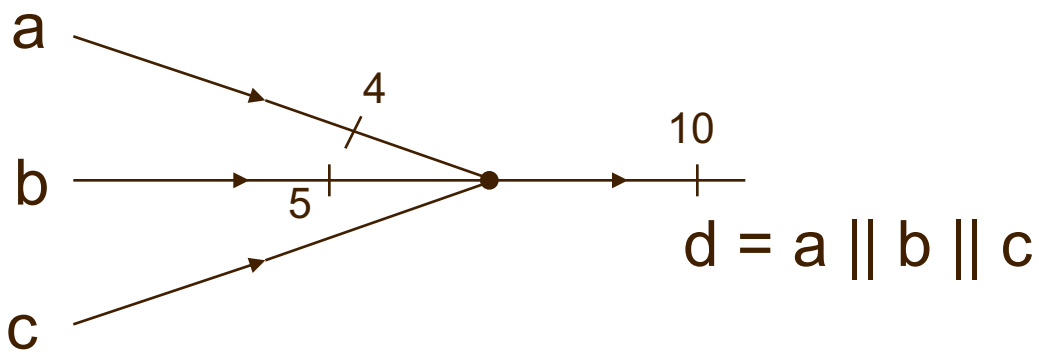
SIGNAL a : STD_LOGIC;



SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);

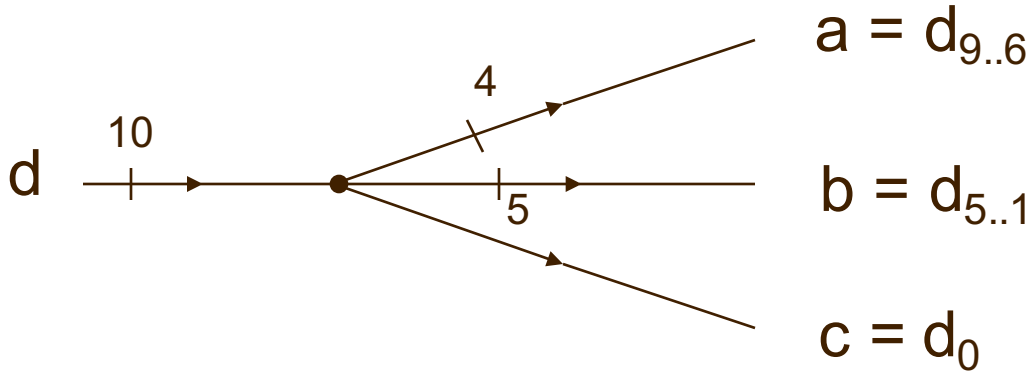


Merging wires and buses

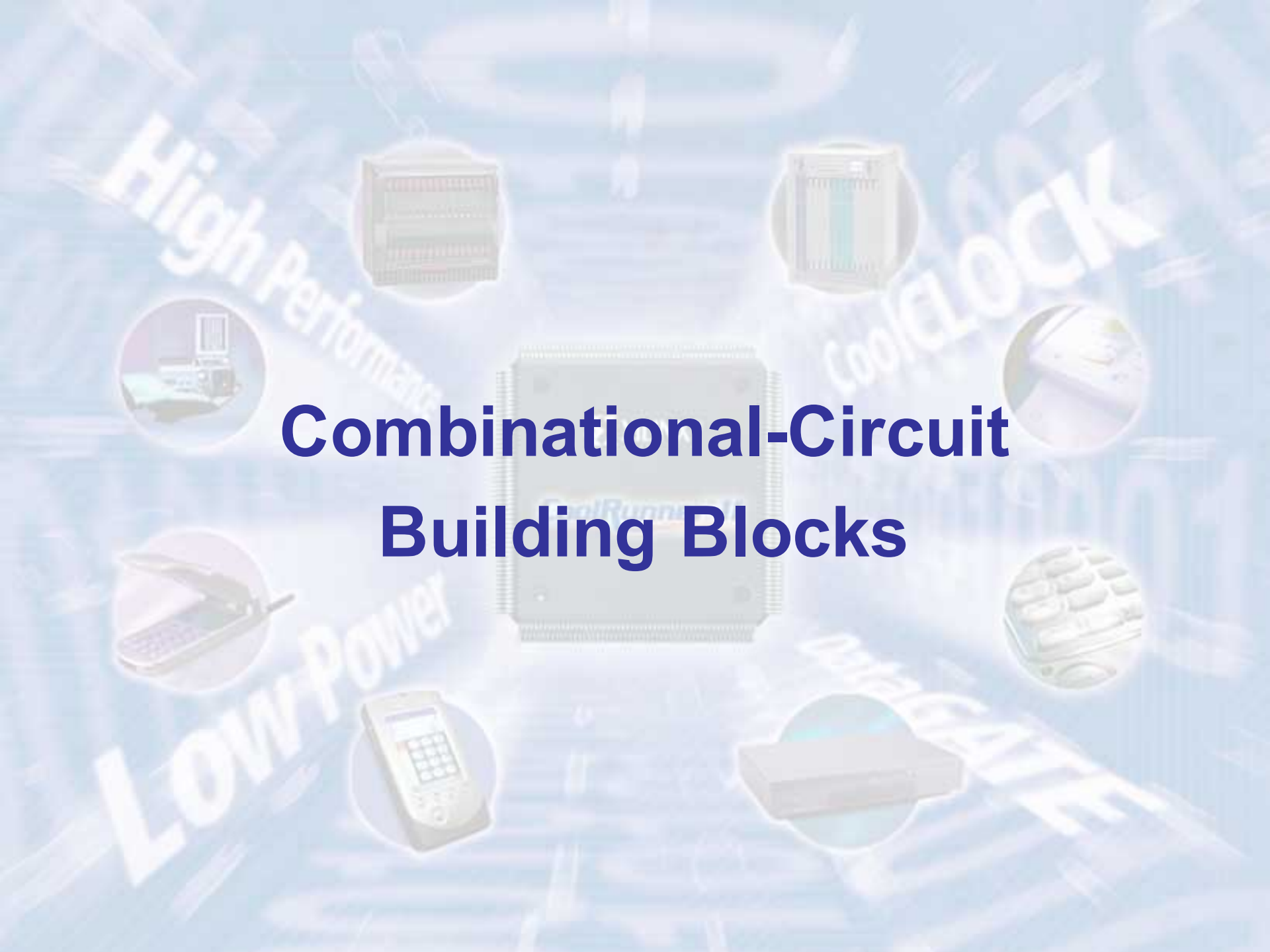



```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

d <= a & b & c;
```

Splitting buses




```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(9 DOWNTO 0);

a <= d(9 downto 6);
b <= d(5 downto 1);
c <= d(0);
```

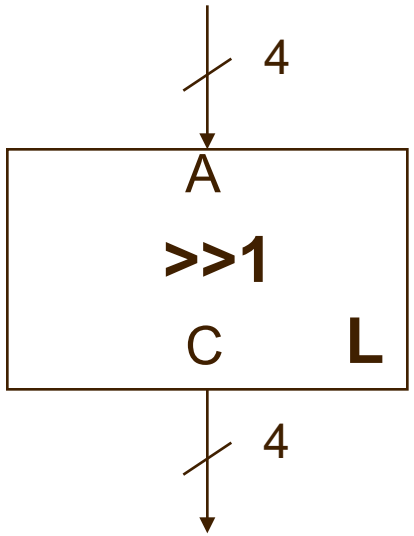


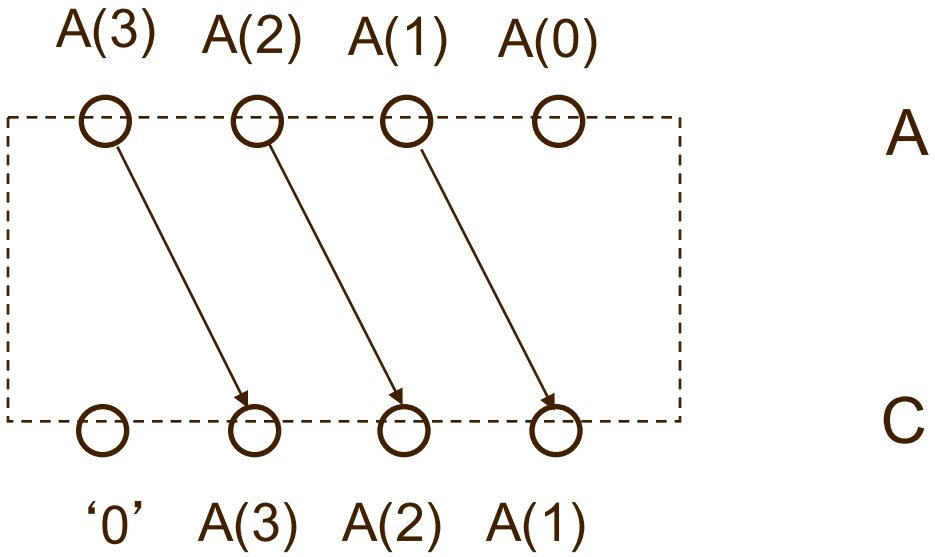
Combinational-Circuit Building Blocks

Fixed Shifters & Rotators

Fixed Logical Shift Right in VHDL

```
SIGNAL A :    STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C:     STD_LOGIC_VECTOR(3 DOWNTO 0);
```

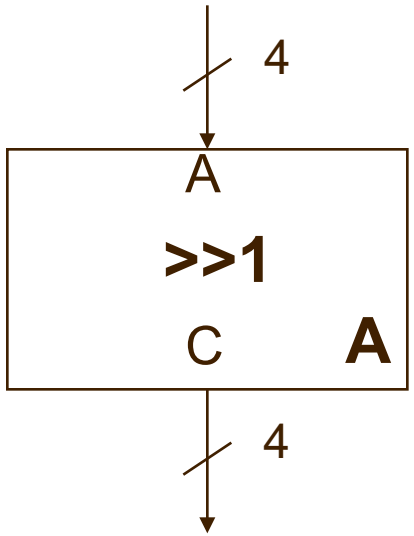


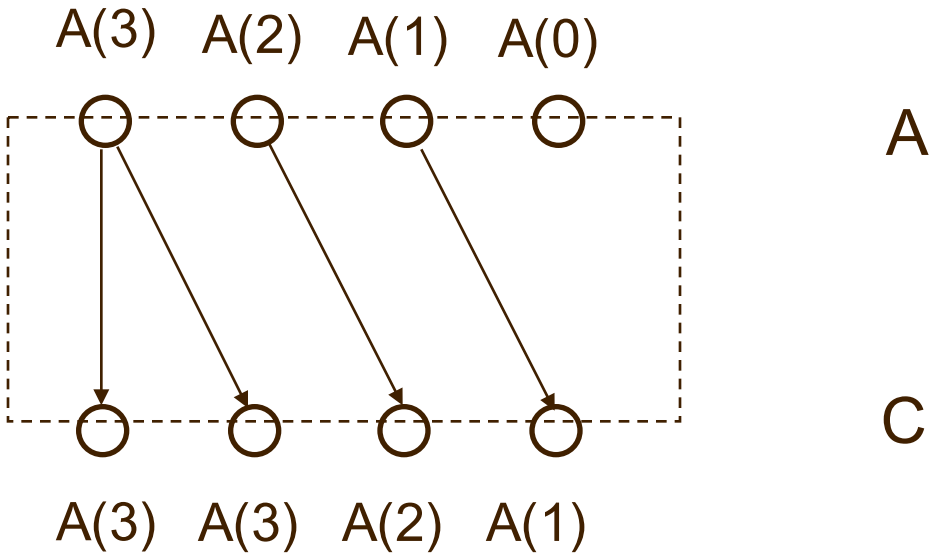


C = '0' & A(3 downto 1);

Fixed Arithmetic Shift Right in VHDL

```
SIGNAL A :    STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C:     STD_LOGIC_VECTOR(3 DOWNTO 0);
```

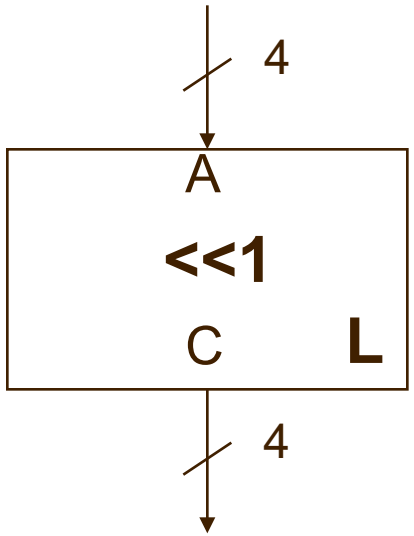


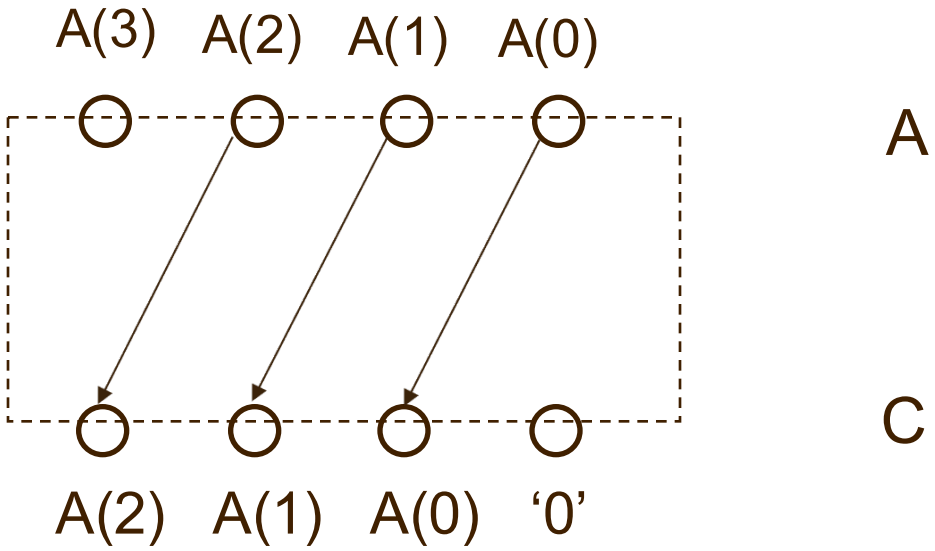


C = A(3) & A(3 downto 1);

Fixed Logical Shift Left in VHDL

```
SIGNAL A :    STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C:     STD_LOGIC_VECTOR(3 DOWNTO 0);
```

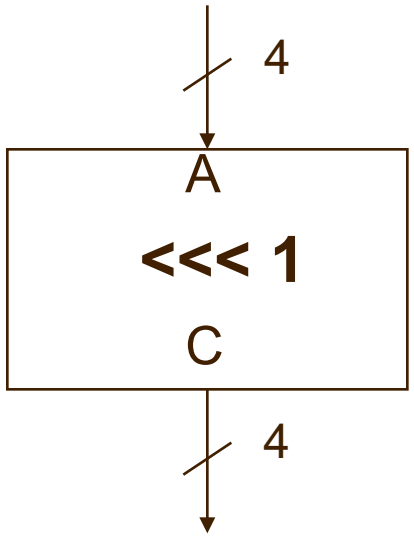


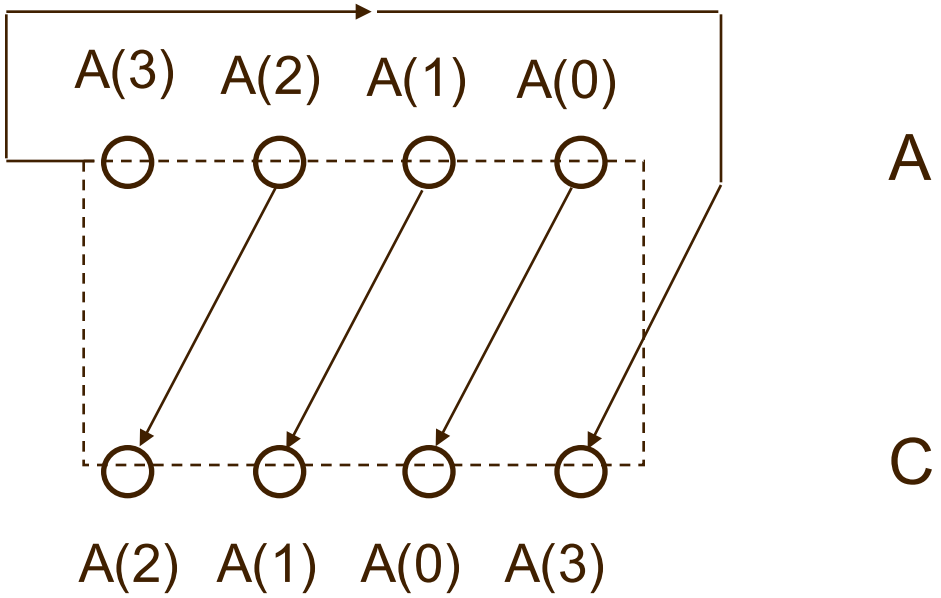


C = A(2 downto 0) & '0';

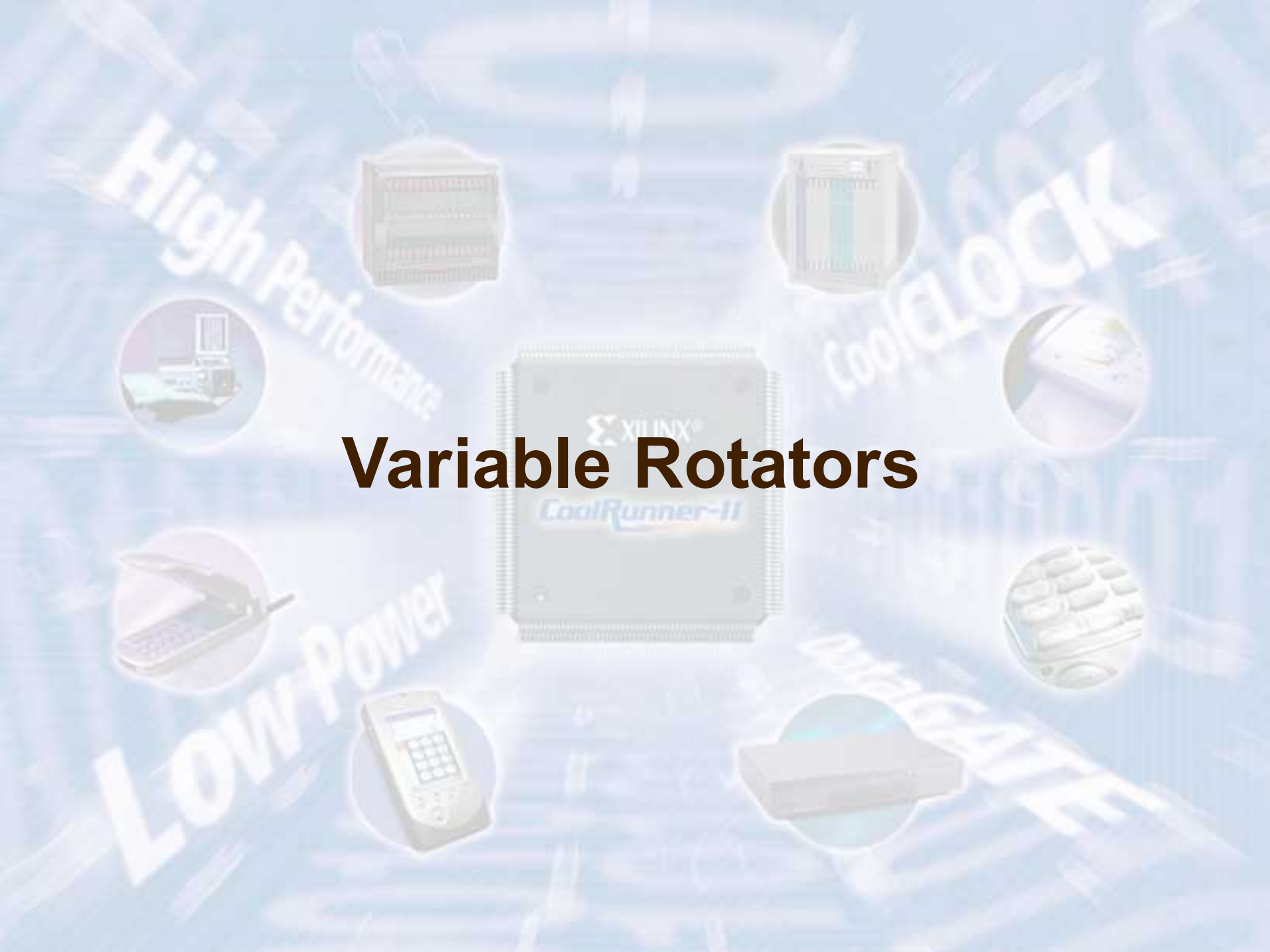
Fixed Rotation Left in VHDL

```
SIGNAL A :    STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C:     STD_LOGIC_VECTOR(3 DOWNTO 0);
```



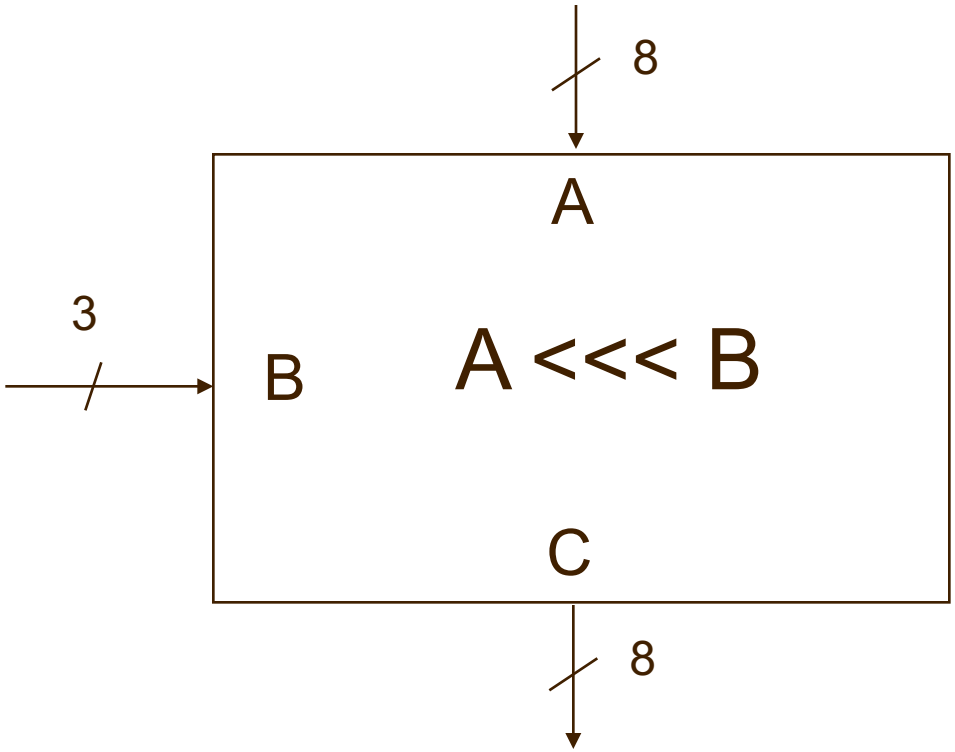


C = A(2 downto 0) & A(3);



Variable Rotators

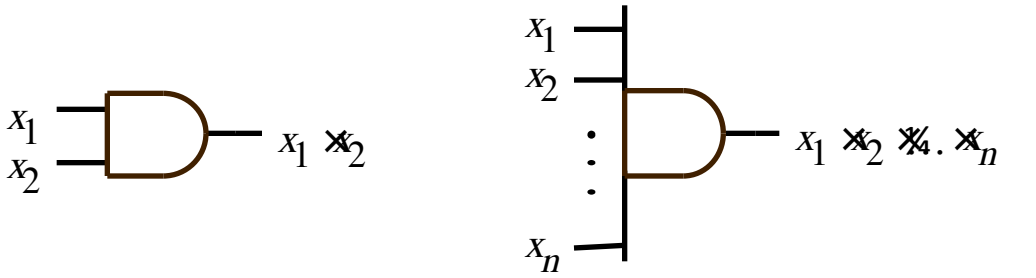
8-bit Variable Rotator Left



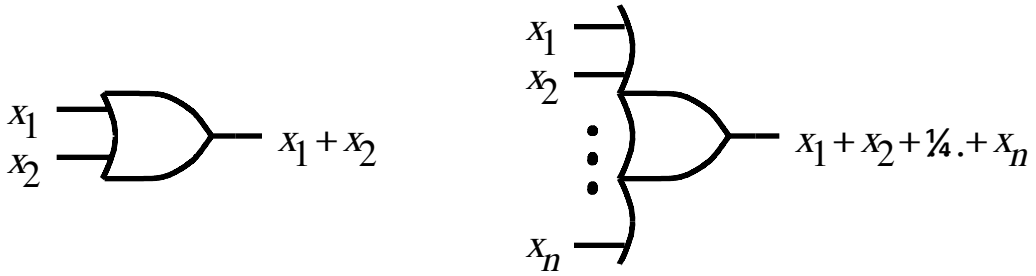
To be covered during one of the future classes

Gates

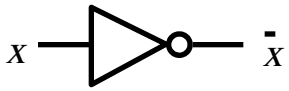
Basic Gates – AND, OR, NOT



(a) AND gates

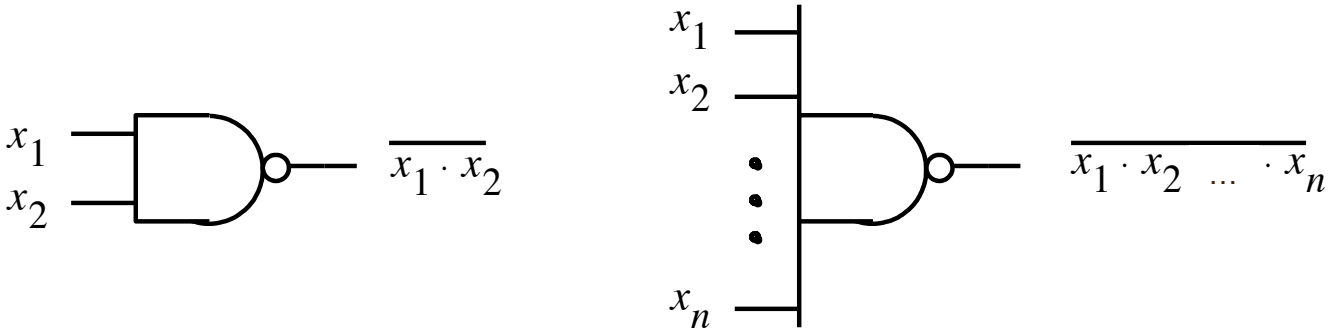


(b) OR gates

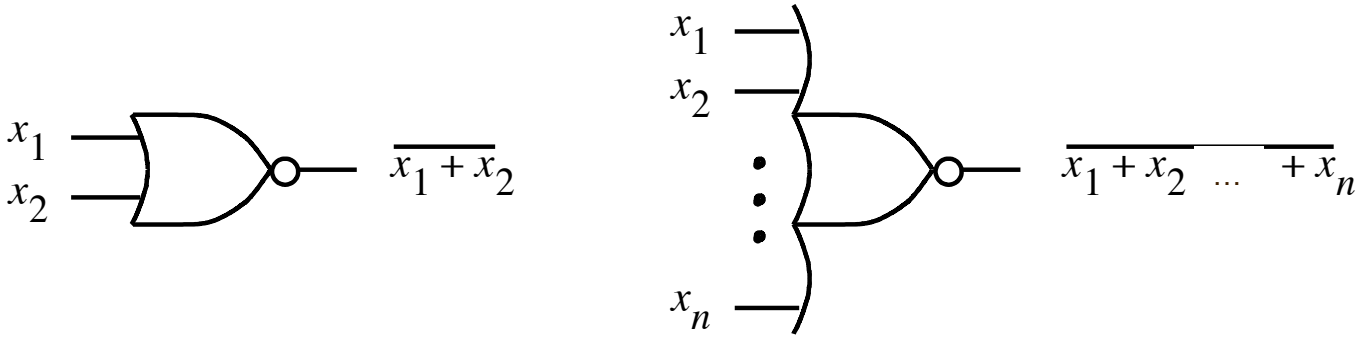


(c) NOT gate

Basic Gates – NAND, NOR

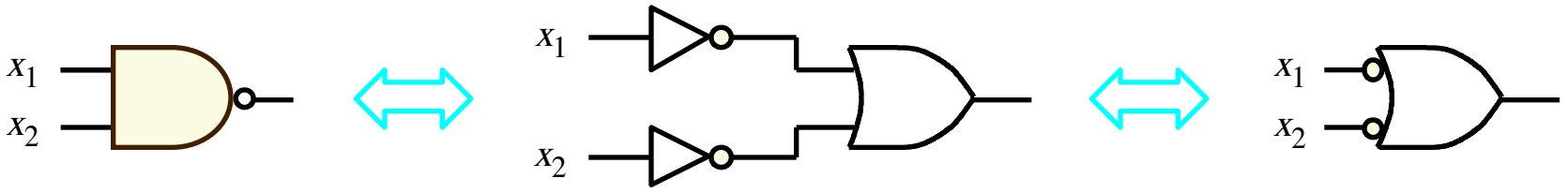


(a) NAND gates

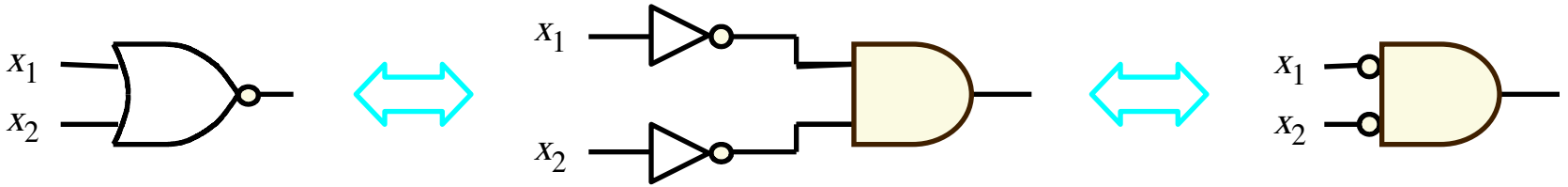


(b) NOR gates

DeMorgan's Theorem and other symbols for NAND, NOR



(a) $\overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2$

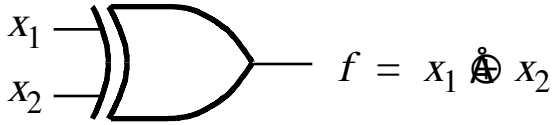


(b) $\overline{x_1 + x_2} = \bar{x}_1 \bar{x}_2$

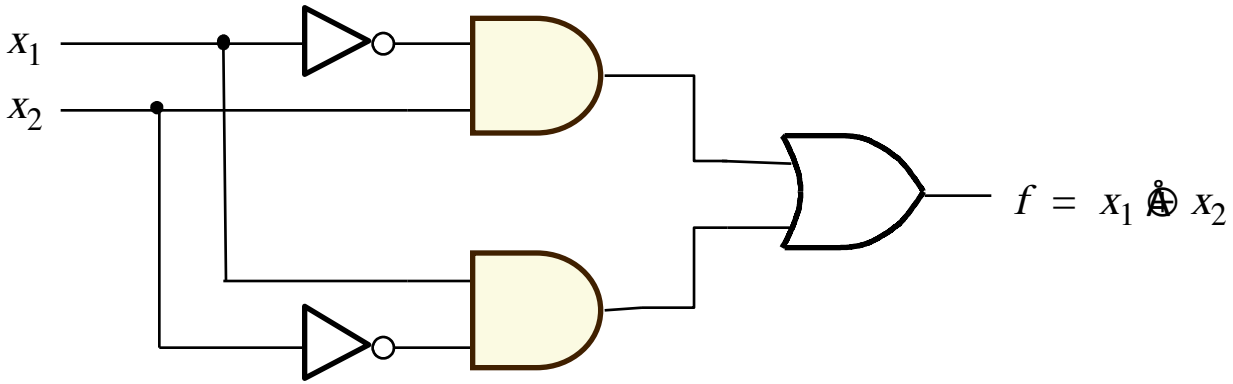
Basic Gates – XOR

x_1	x_2	$f = x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

(a) Truth table



(b) Graphical symbol

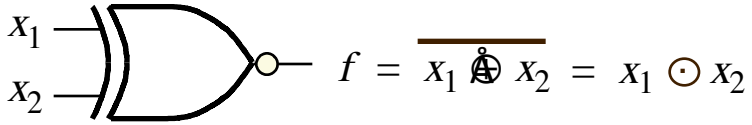


(c) Sum-of-products implementation

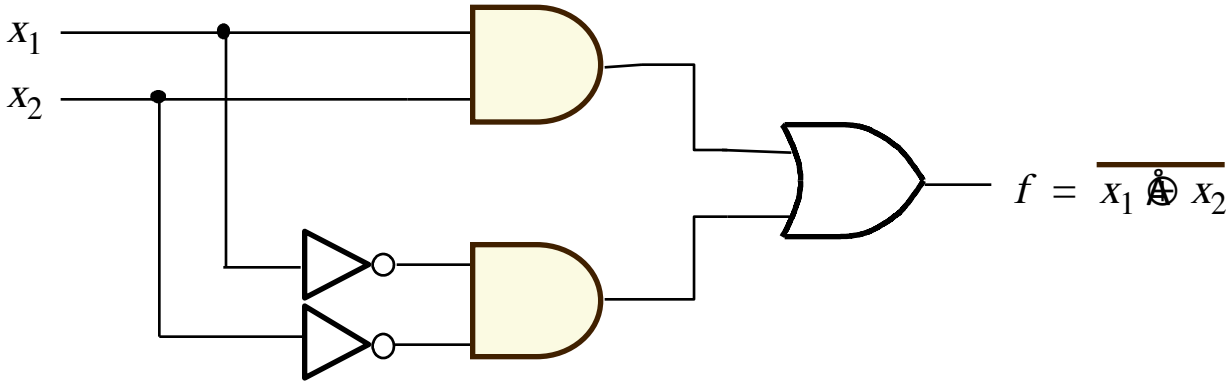
Basic Gates – XNOR

x_1	x_2	$f = \overline{x_1 \oplus x_2}$
0	0	1
0	1	0
1	0	0
1	1	1

(a) Truth table

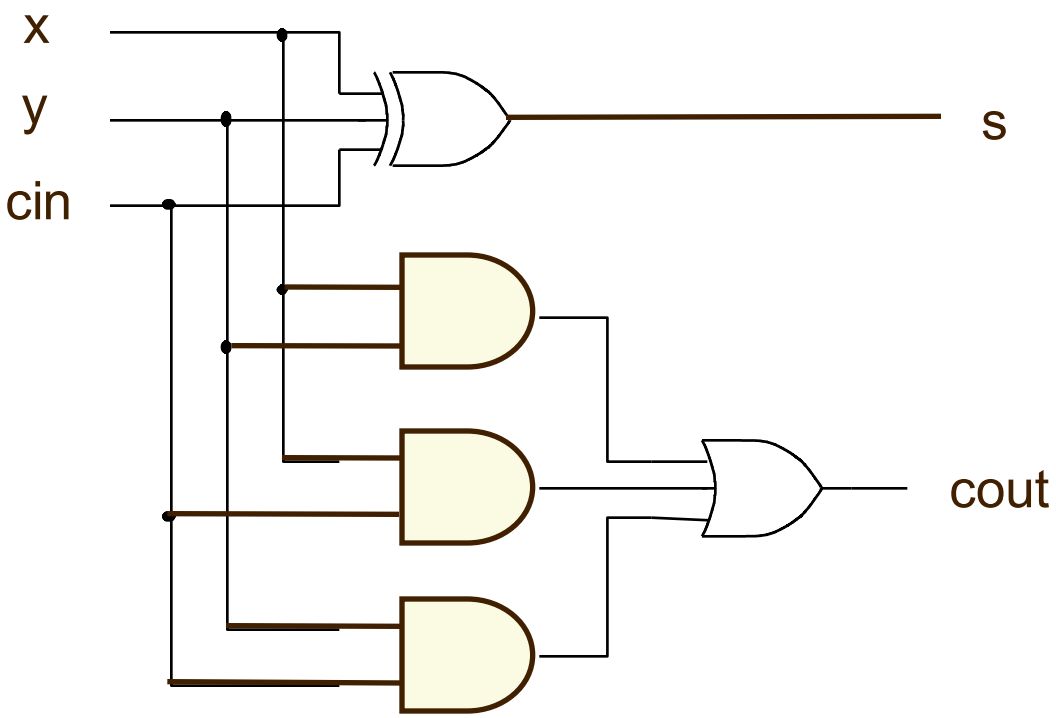


(b) Graphical symbol



(c) Sum-of-products implementation

Data-flow VHDL: Example



Data-flow VHDL: Example (1)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY fulladd IS
        PORT ( x      : IN     STD_LOGIC ;
               y      : IN     STD_LOGIC ;
               cin    : IN     STD_LOGIC ;
               s      : OUT    STD_LOGIC ;
               cout   : OUT    STD_LOGIC ) ;
END fulladd ;
```

Data-flow VHDL: Example (2)

```
ARCHITECTURE dataflow OF fulladd IS
BEGIN
        s     <=  x XOR y XOR cin ;
        cout  <=  (x AND y) OR (cin AND x) OR (cin AND y) ;
END dataflow ;
```

Logic Operators

- Logic operators

- Logic operators precedence

No Implied Precedence

Wanted: y **= ab + cd**

Incorrect

y <= a **and** b **or** c **and** d ;

equivalent to

y <= ((a **and** b) **or** c) **and** d ;

equivalent to

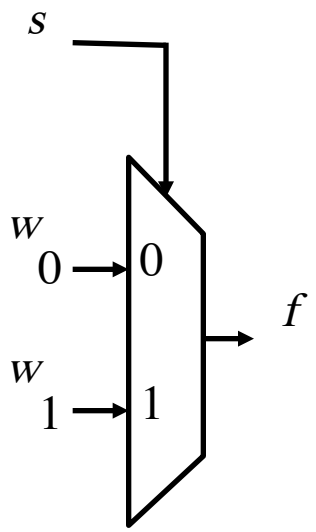
y = (ab + c)d

Correct

y <= (a **and** b) **or** (c **and** d) ;

Multiplexers

2-to-1 Multiplexer



(a) Graphical symbol

s	f
0	w_0
1	w_1

(b) Truth table

VHDL: **f <= w0 WHEN s = '0' ELSE w1 ;**

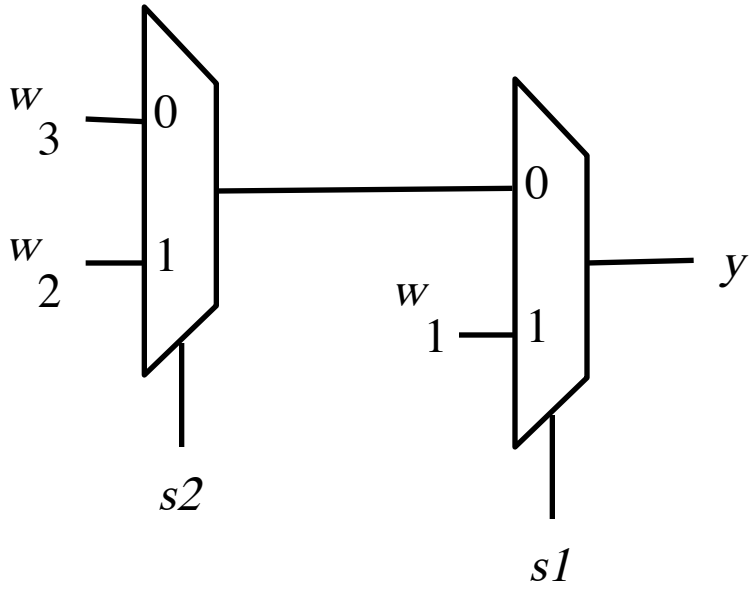
or

f <= w1 WHEN s = '1' ELSE w0 ;

VHDL code for a 2-to-1 Multiplexer Entity

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT ( w0, w1, s  : IN     STD_LOGIC ;
                 f    : OUT    STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE dataflow OF mux2to1 IS
BEGIN
    f <= w0 WHEN s = '0' ELSE w1 ;
END dataflow ;
```

Cascade of two multiplexers



VHDL:

```
f <= w1  WHEN s1 = '1' ELSE
     w2  WHEN s2 = '1' ELSE
     w3 ;
```

VHDL design entity implementing a cascade of two multiplexers

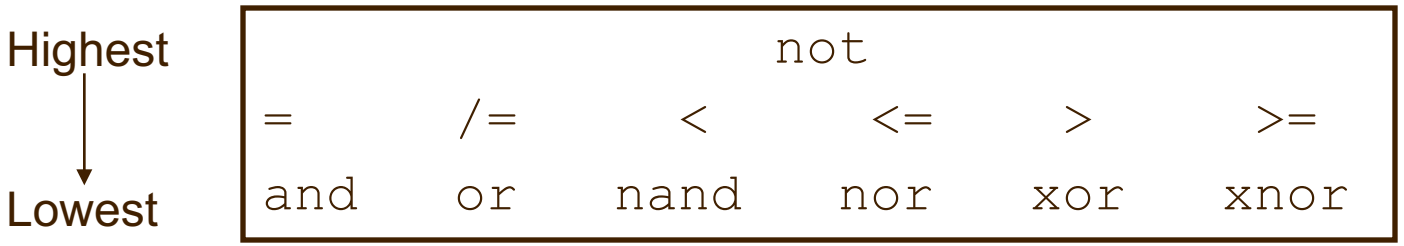
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux_cascade IS
    PORT (  w1, w2, w3: IN  STD_LOGIC ;
            s1, s2      : IN  STD_LOGIC ;
            f           : OUT  STD_LOGIC ) ;
END mux_cascade ;

ARCHITECTURE dataflow OF mux2to1 IS
BEGIN
    f <= w1 WHEN s1 = '1' ELSE
         w2  WHEN s2 = '1' ELSE
         w3 ;
END dataflow ;
```

Operators

- Relational operators

```
=       /=       <       <=      >       >=
```

- Logic and relational operators precedence



Priority of logic and relational operators

compare a = bc

Incorrect

… when a = b **and** c else …

equivalent to

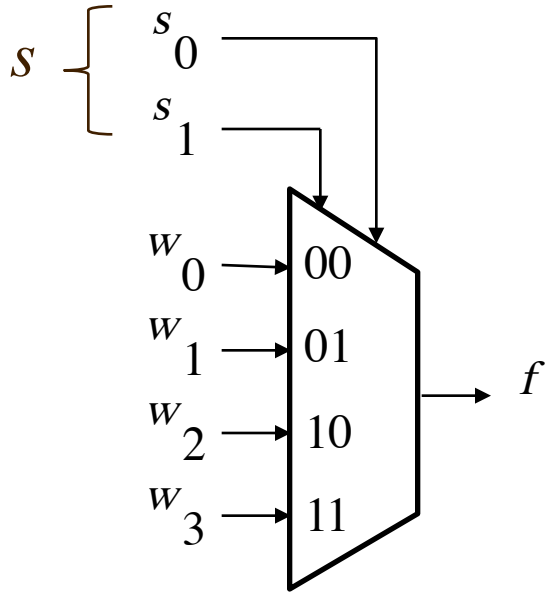
… when (a = b) **and** c else …

Correct

… when a = (b **and** c) else …

4-to-1 Multiplexer

(a) Graphic symbol



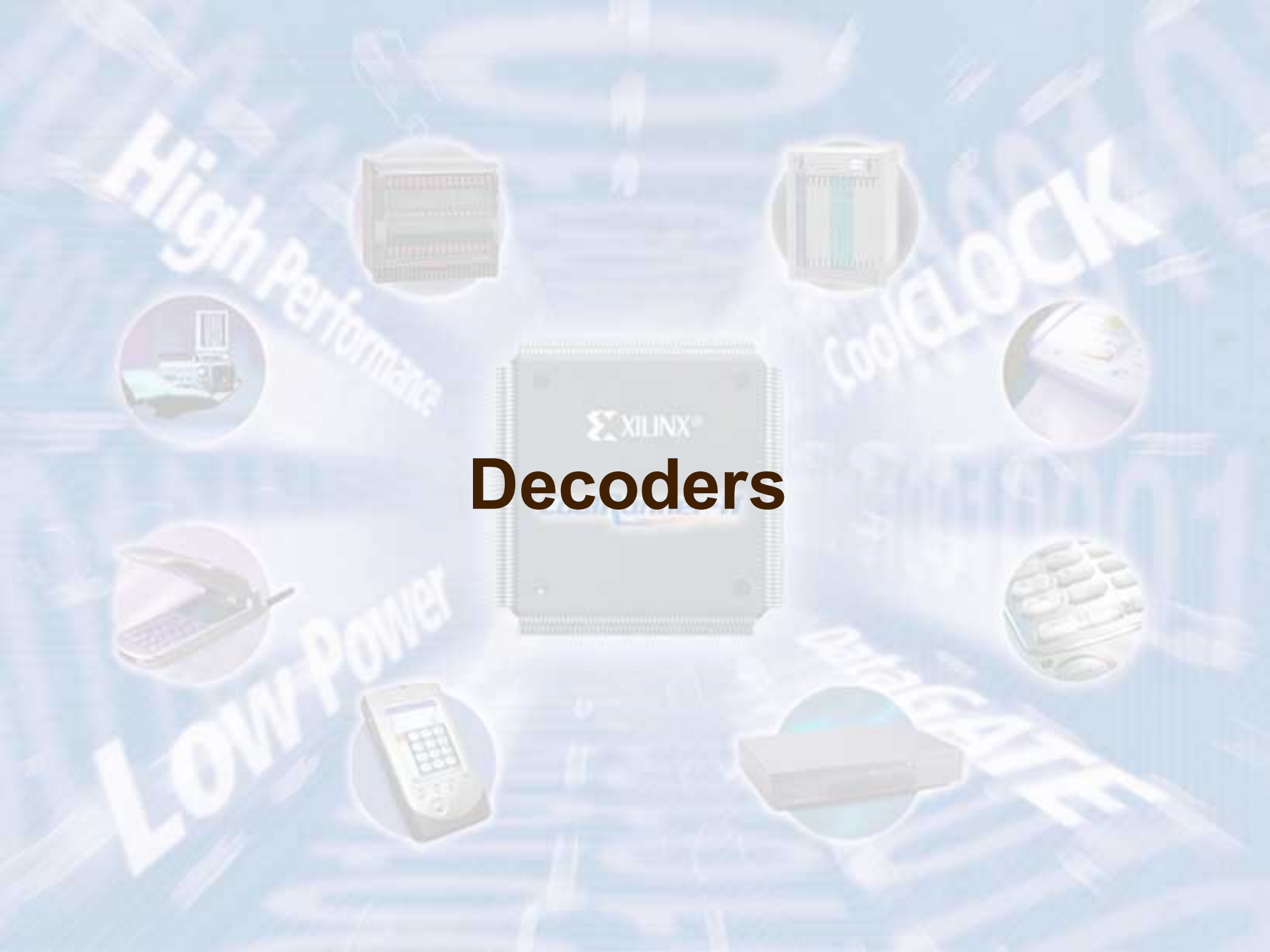
(b) Truth table

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

```
WITH s SELECT
        f <= w0 WHEN "00",
             w1 WHEN "01",
             w2 WHEN "10",
             w3 WHEN OTHERS ;
```

VHDL code for a 4-to-1 Multiplexer entity

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux4to1 IS
      PORT (   w0, w1, w2, w3     : IN      STD_LOGIC ;
               s                  : IN      STD_LOGIC_VECTOR(1 DOWNTO 0) ;
               f                  : OUT     STD_LOGIC ) ;
END mux4to1 ;

ARCHITECTURE dataflow OF mux4to1 IS
BEGIN
      WITH s SELECT
            f <= w0 WHEN "00",
                 w1 WHEN "01",
                 w2 WHEN "10",
                 w3 WHEN OTHERS ;
END dataflow ;
```



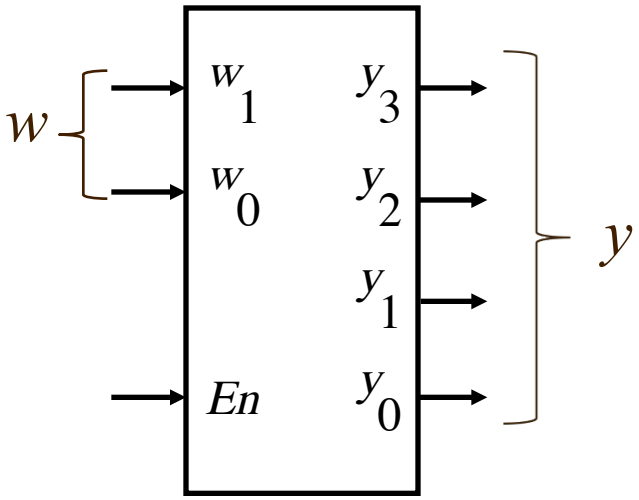
Decoders

2-to-4 Decoder

(a) Truth table

En	w_1	w_0	y_3	y_2	y_1	y_0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	x	x	0	0	0	0

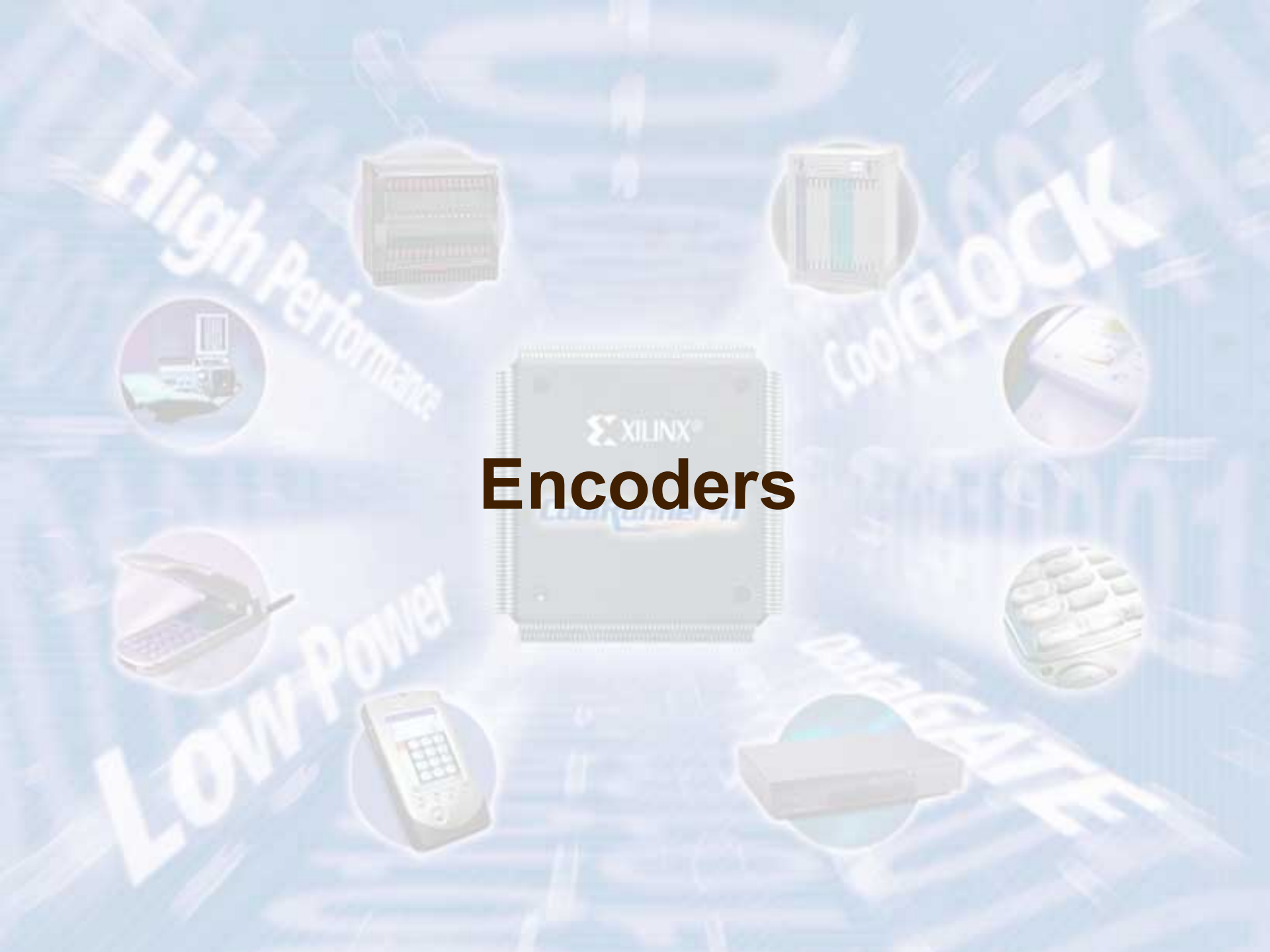
(b) Graphical symbol




```
Enw <= En & w ;
WITH Enw SELECT
    y <= "0001" WHEN "100",
         "0010" WHEN "101",
         "0100" WHEN "110",
         "1000" WHEN "111",
         "0000" WHEN OTHERS ;
```

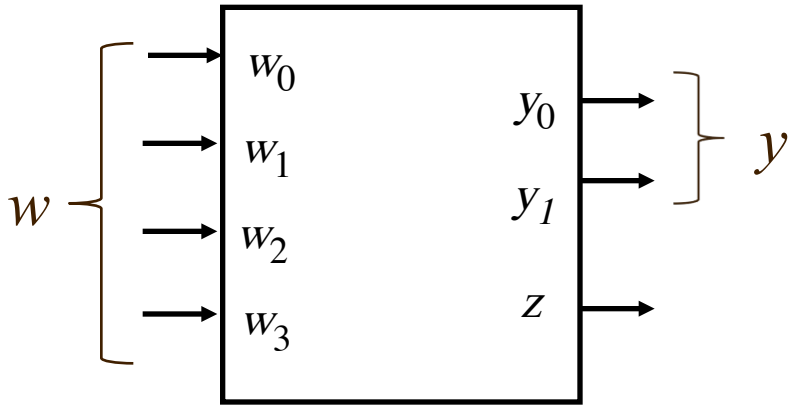
VHDL code for a 2-to-4 Decoder entity

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY dec2to4 IS
      PORT (    w     : IN      STD_LOGIC_VECTOR(1 DOWNTO 0) ;
                En    : IN      STD_LOGIC ;
                y     : OUT     STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END dec2to4 ;

ARCHITECTURE  dataflow OF dec2to4 IS
      SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0) ;
BEGIN
      Enw <= En & w ;
      WITH Enw SELECT
            y <= "0001" WHEN "100",
                 "0010" WHEN "101",
                 "0100" WHEN "110",
                 "1000" WHEN "111",
                 "0000" WHEN OTHERS ;
END dataflow ;
```



Encoders

Priority Encoder



w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	-	0	1	1
0	1	-	-	1	0	1
1	-	-	-	1	1	1

```
y <= "11" WHEN w(3) = '1' ELSE
     "10" WHEN w(2) = '1' ELSE
     "01" WHEN w(1) = '1' ELSE
     "00" ;
```

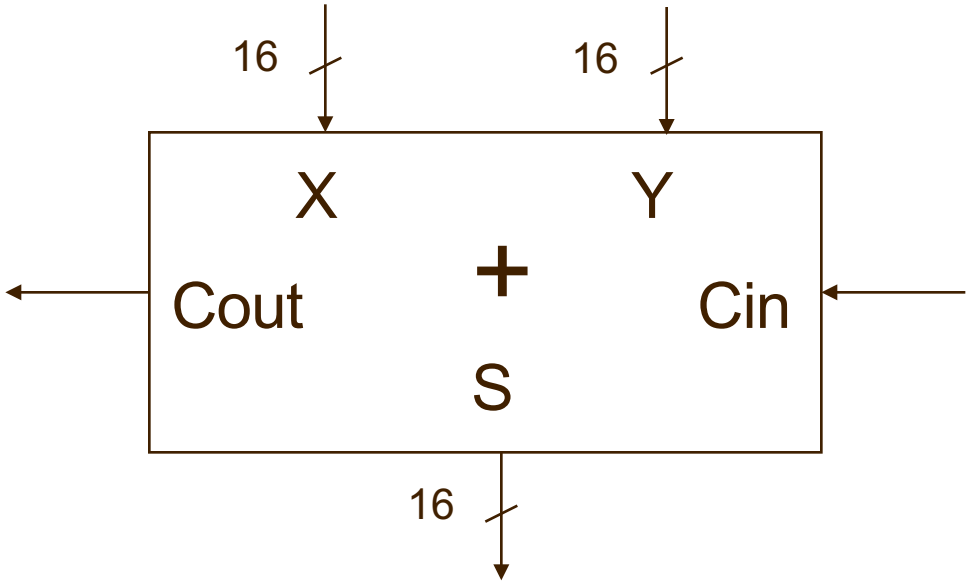
```
z <= '0' WHEN w = "0000" ELSE '1' ;
```

VHDL code for a Priority Encoder entity

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all;

ENTITY priority IS
    PORT ( w   : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           y   : OUT   STD_LOGIC_VECTOR(1 DOWNTO 0) ;
           z   : OUT   STD_LOGIC ) ;
END priority ;

ARCHITECTURE dataflow OF priority IS
BEGIN
    y <=  "11" WHEN w(3) = '1' ELSE
          "10" WHEN w(2) = '1' ELSE
          "01" WHEN w(1) = '1' ELSE
          "00" ;
    z <= '0' WHEN w = "0000" ELSE '1' ;
END dataflow ;
```

Adders

16-bit Unsigned Adder



Operations on Unsigned Numbers

For operations on unsigned numbers

USE ieee.std_logic_unsigned.all
and
signals of the type
STD_LOGIC_VECTOR

OR

USE ieee.numeric_std.all
and
signals of the type
UNSIGNED
and conversion functions:
std_logic_vector(), unsigned()

VHDL code for a 16-bit Unsigned Adder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY adder16 IS
      PORT (   Cin              : IN     STD_LOGIC ;
               X                : IN     STD_LOGIC_VECTOR(15 DOWNTO 0) ;
               Y                : IN     STD_LOGIC_VECTOR(15 DOWNTO 0) ;
               S                : OUT    STD_LOGIC_VECTOR(15 DOWNTO 0) ;
               Cout             : OUT    STD_LOGIC ) ;
END adder16 ;

ARCHITECTURE dataflow OF adder16 IS
      SIGNAL Sum : STD_LOGIC_VECTOR(16 DOWNTO 0) ;
BEGIN
      Sum <= ('0' & X) + Y + Cin ;
      S <= Sum(15 DOWNTO 0) ;
      Cout <= Sum(16) ;
END dataflow ;
```

Addition of Unsigned Numbers (1)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.numeric_std.all ;


ENTITY adder16 IS
      PORT ( Cin          : IN     STD_LOGIC ;
             X            : IN     STD_LOGIC_VECTOR(15 DOWNTO 0) ;
             Y            : IN     STD_LOGIC_VECTOR(15 DOWNTO 0) ;
             S            : OUT    STD_LOGIC_VECTOR(15 DOWNTO 0) ;
             Cout         : OUT    STD_LOGIC ) ;
END adder16 ;
```

Addition of Unsigned Numbers (2)

```
ARCHITECTURE  dataflow OF adder16 IS
        SIGNAL Xu : UNSIGNED(15 DOWNTO 0);
        SIGNAL Yu: UNSIGNED(15 DOWNTO 0);
        SIGNAL Su : UNSIGNED(16 DOWNTO 0) ;
BEGIN
        Xu <= unsigned(X);
        Yu <= unsigned(Y);
        Su <= ('0' & Xu) + Yu + unsigned(std_logic_vector' ('0' & Cin)) ;
        S <= std_logic_vector(Su(15 DOWNTO 0)) ;
        Cout <= Su(16) ;
END dataflow ;
```

Addition of Unsigned Numbers (3)

```
ARCHITECTURE  dataflow OF adder16 IS
      signal Sum: STD_LOGIC_VECTOR(16 DOWNTO 0) ;
BEGIN
      Sum <= std_logic_vector( unsigned('0' & X) + unsigned(Y)
                                    + unsigned(std_logic_vector' ('0' & Cin)) ) ;
      S <= Sum(15 downto 0);
      Cout <= Sum(16) ;
END dataflow ;
```

Operations on Signed Numbers

For operations on signed numbers

USE ieee.numeric_std.all,
signals of the type
SIGNED,
and conversion functions:
std_logic_vector(), signed()

OR

USE ieee.std_logic_signed.all
and signals of the type
STD_LOGIC_VECTOR

Signed and Unsigned Types

Behave exactly **like**

STD_LOGIC_VECTOR

plus, they determine whether a given vector should be treated as a signed or unsigned number.

Require

USE ieee.numeric_std.all;


High Performance
Low Power
Multipliers

Unsigned vs. Signed Multiplication

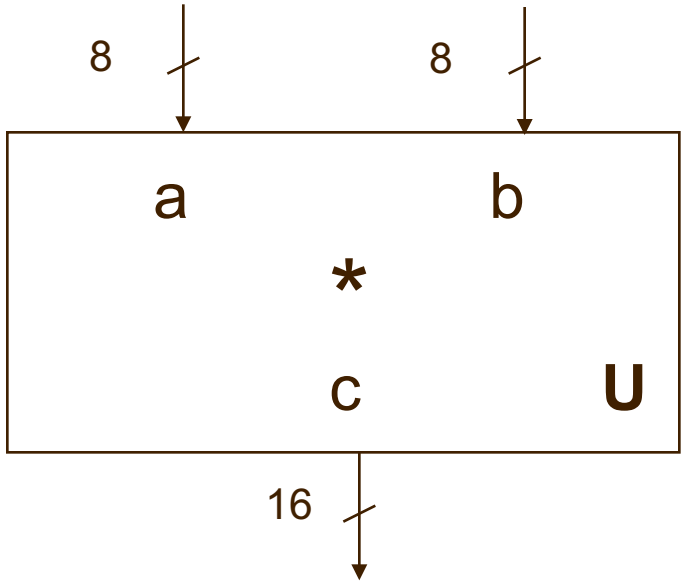
Unsigned

```
    1111        15
  x 1111      x 15
--------    ------
11100001       225
```

Signed

```
    1111        -1
  x 1111      x -1
--------    ------
00000001         1
```

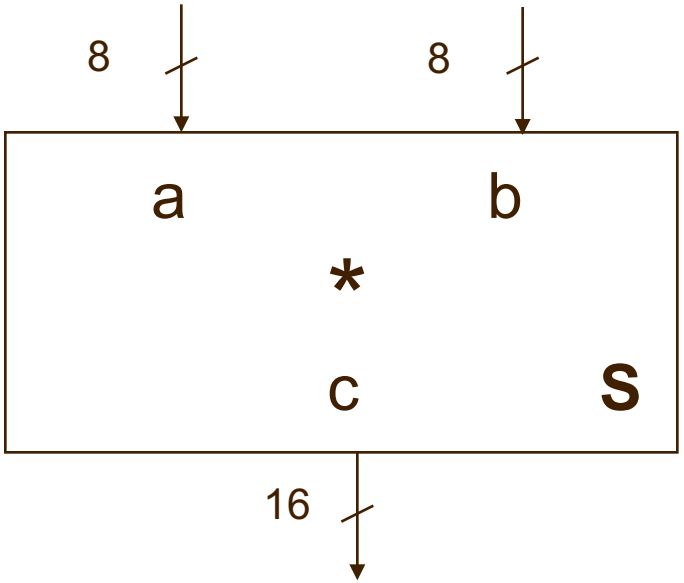
8x8-bit Unsigned Multiplier



Multiplication of unsigned numbers

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all ;

entity multiply is
      port(
            a : in STD_LOGIC_VECTOR(7 downto 0);
            b : in STD_LOGIC_VECTOR(7 downto 0);
            c : out STD_LOGIC_VECTOR(15 downto 0)
          );
end multiply;

architecture dataflow of multiply is
begin
    c <= a * b;
end dataflow;
```

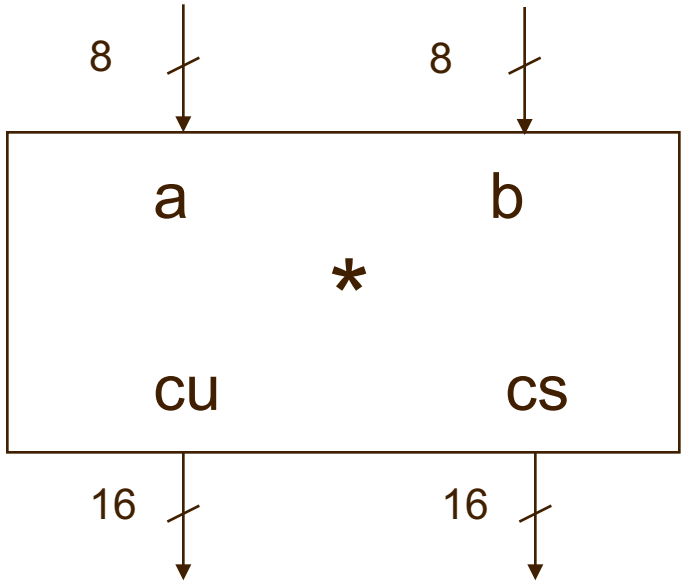
8x8-bit Signed Multiplier



Multiplication of signed numbers

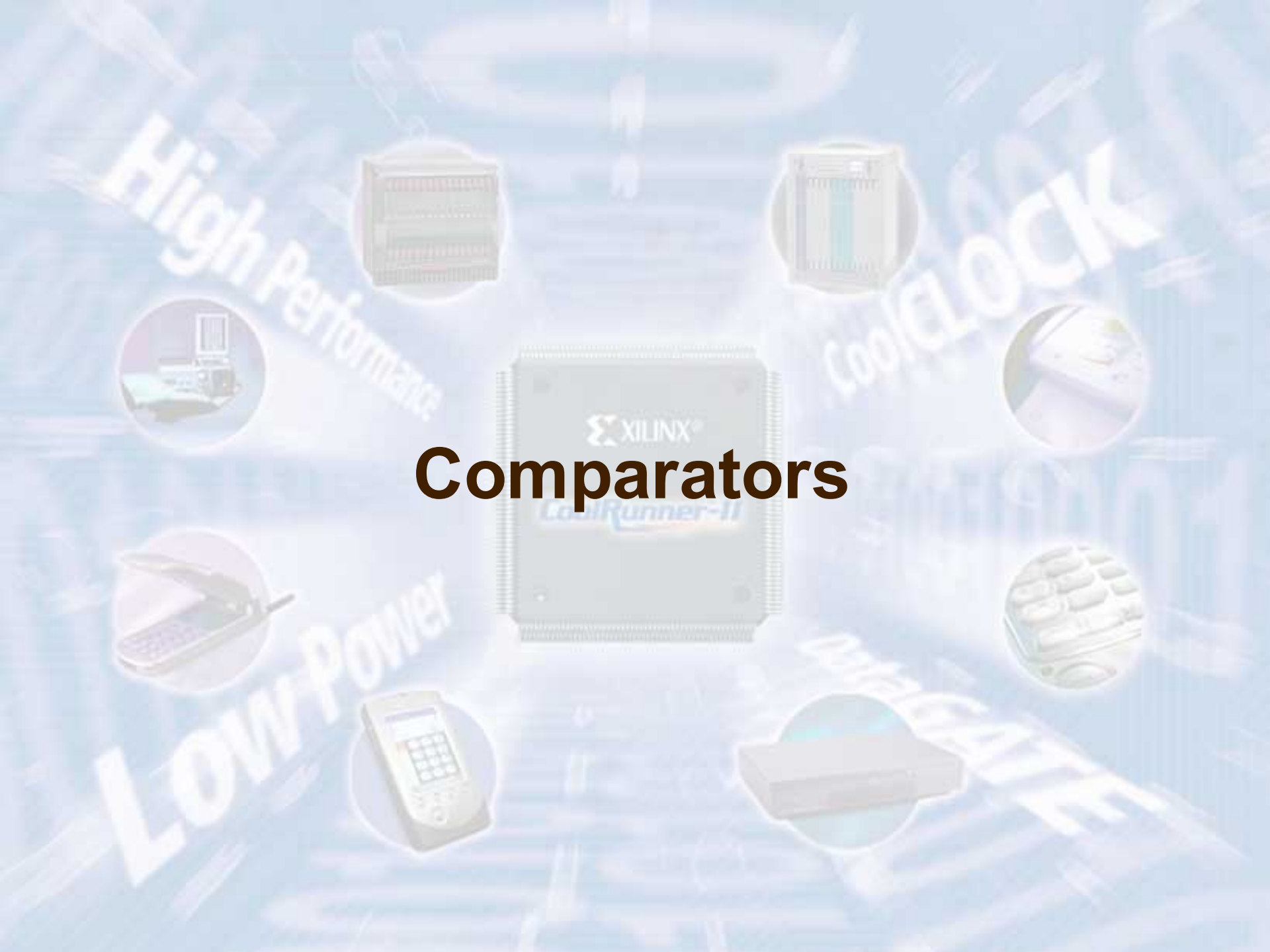
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all ;

entity multiply is
      port(
            a : in STD_LOGIC_VECTOR(7 downto 0);
            b : in STD_LOGIC_VECTOR(7 downto 0);
            c : out STD_LOGIC_VECTOR(15 downto 0)
          );
end multiply;

architecture dataflow of multiply is
begin
    c <= a * b;
end dataflow;
```

8x8-bit Unsigned and Signed Multiplier



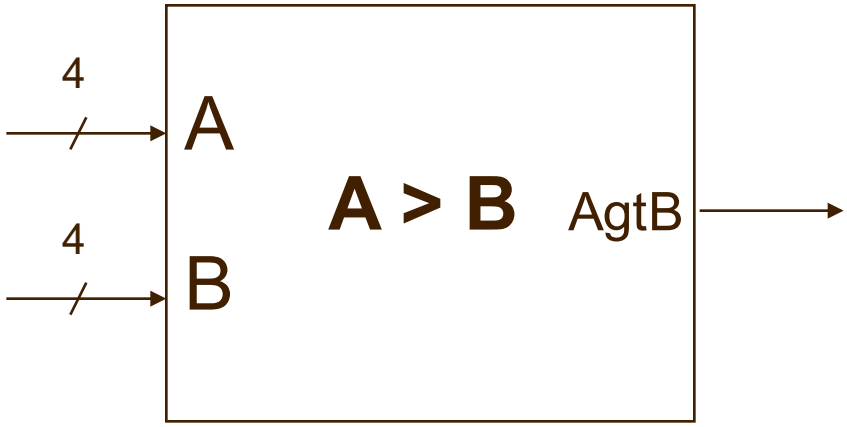
Multiplication of signed and unsigned numbers

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all ;

entity multiply is
      port(
            a : in STD_LOGIC_VECTOR(7 downto 0);
            b : in STD_LOGIC_VECTOR(7 downto 0);
            cu : out STD_LOGIC_VECTOR(15 downto 0);
            cs : out STD_LOGIC_VECTOR(15 downto 0)
         );
end multiply;

architecture dataflow of multiply is
begin

-- signed multiplication
     cs <= STD_LOGIC_VECTOR(SIGNED(a)*SIGNED(b));

-- unsigned multiplication
     cu <= STD_LOGIC_VECTOR(UNSIGNED(a)*UNSIGNED(b));
end dataflow;
```



Comparators

4-bit Number Comparator



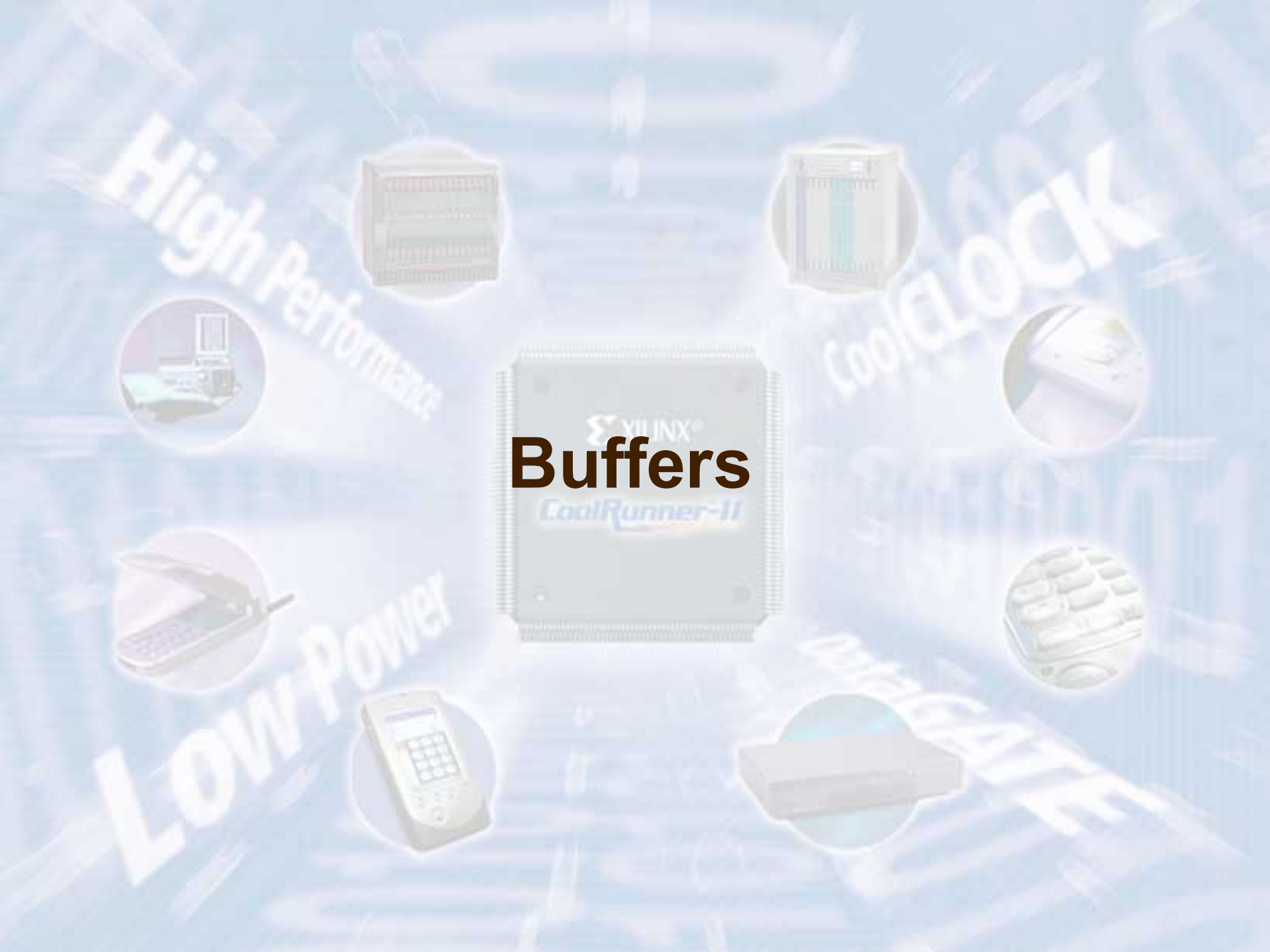

```
AgtB <= '1' WHEN A > B ELSE '0' ;
```

VHDL code for a 4-bit **Unsigned** Number Comparator entity

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY compare IS
     PORT (  A, B              : IN     STD_LOGIC_VECTOR(3  DOWNTO 0) ;
             AgtB              : OUT    STD_LOGIC ) ;
END compare ;

ARCHITECTURE  dataflow OF compare IS
BEGIN
     AgtB <= '1' WHEN A > B ELSE '0' ;
END dataflow ;
```

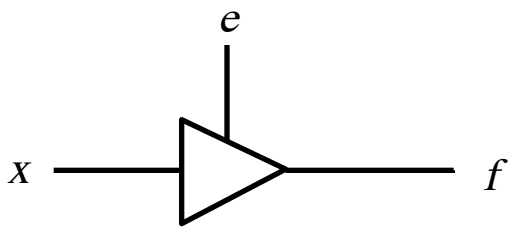
VHDL code for a 4-bit **Signed** Number Comparator entity

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all ;

ENTITY compare IS
     PORT (  A, B              : IN     STD_LOGIC_VECTOR(3  DOWNTO 0) ;
                 AgtB          : OUT    STD_LOGIC ) ;
END compare ;

ARCHITECTURE  dataflow OF compare IS
BEGIN
     AgtB  <= '1' WHEN  A > B ELSE  '0' ;
END dataflow ;
```

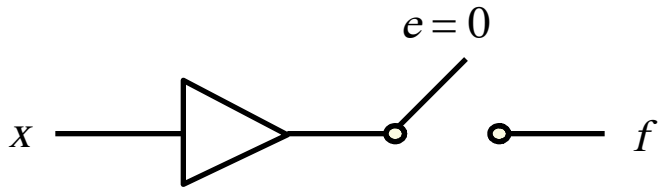


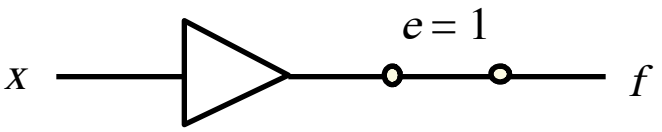
Buffers

Tri-state Buffer



(a) A tri-state buffer



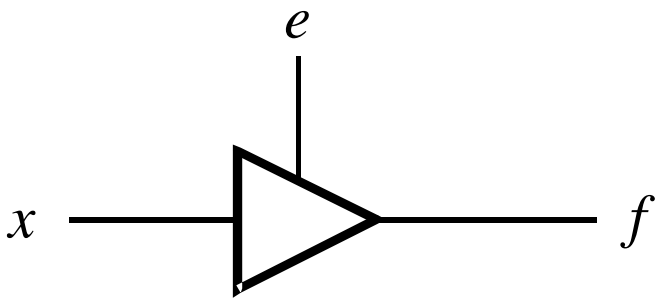


(b) Equivalent circuit

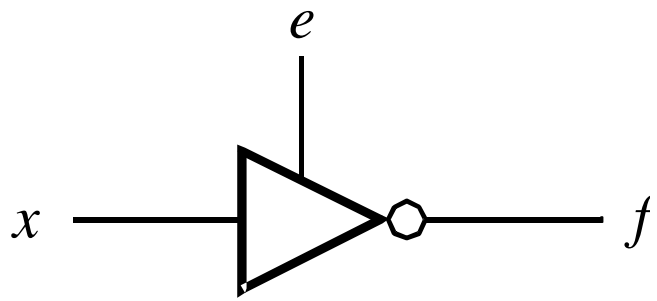
e	x	f
0	0	Z
0	1	Z
1	0	0
1	1	1

(c) Truth table

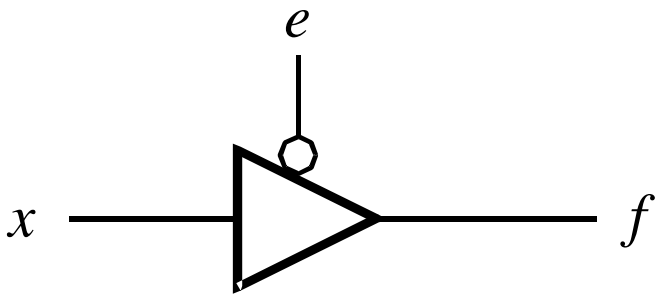
Four types of Tri-state Buffers



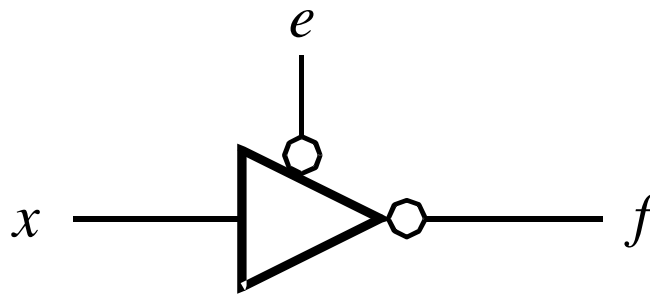
f <= x WHEN (e = '1') ELSE 'Z';



f <= not x WHEN (e = '1') ELSE 'Z';



f <= x WHEN (e = '0') ELSE 'Z';



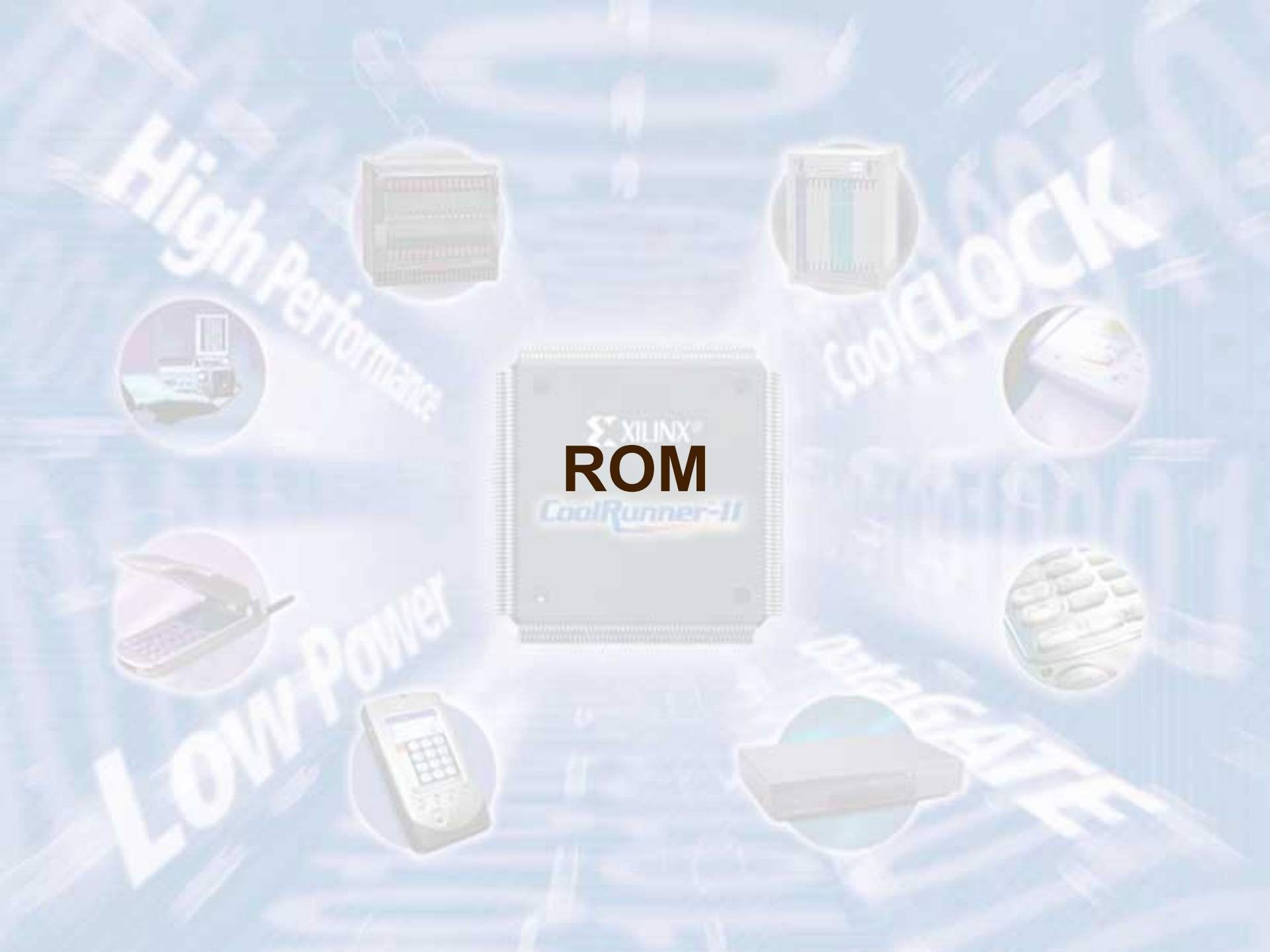
f <= not x WHEN (e = '0') ELSE 'Z';

Tri-state Buffer entity (1)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY tri_state IS
   PORT ( e:    IN STD_LOGIC;
                x:  IN STD_LOGIC;
                f: OUT STD_LOGIC
               );
END tri_state;
```

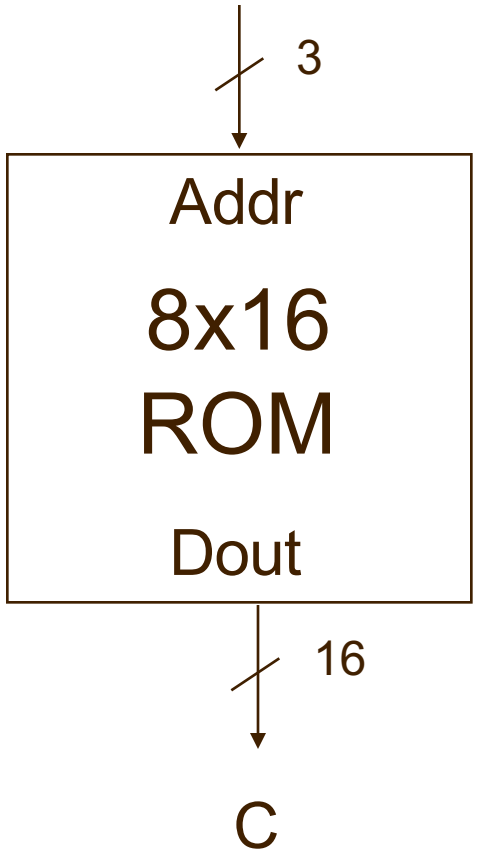
Tri-state Buffer entity (2)

```
ARCHITECTURE dataflow OF tri_state IS
BEGIN
    f <= x WHEN (e = '1') ELSE   'Z';
END dataflow;
```



ROM

ROM 8x16 example (1)



ROM 8x16 example (2)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY rom IS

        PORT (
                Addr : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
                Dout : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
            );

END rom;
```

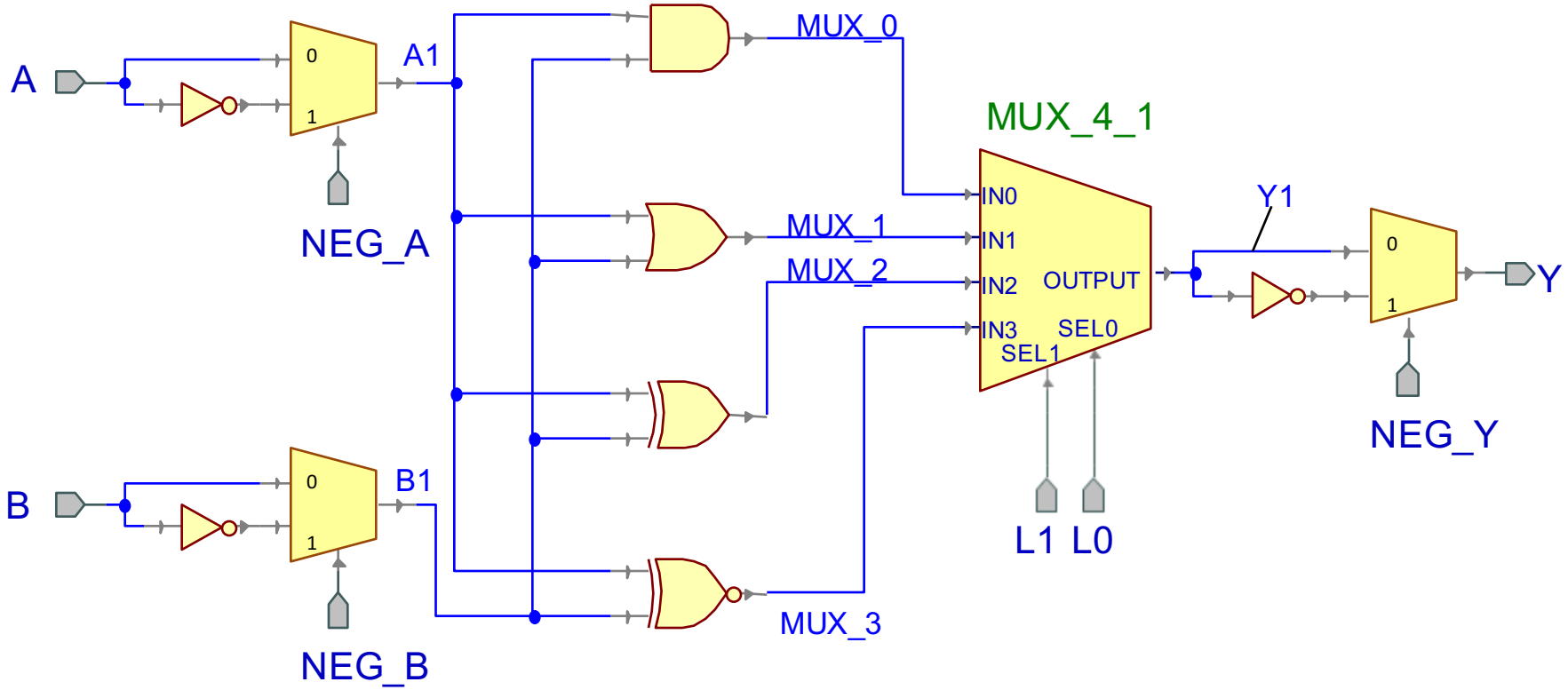
ROM 8x16 example (3)

```
ARCHITECTURE dataflow OF rom IS
SIGNAL temp: INTEGER RANGE 0 TO 7;
TYPE vector_array IS ARRAY (0 to 7) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
CONSTANT memory : vector_array :=
                                ( X”800A",
                                  X"D459",
                                  X"A870",
                                  X"7853",
                                  X"650D",
                                  X"642F",
                                  X"F742",
                                  X"F548");
BEGIN

    temp <= to_integer(unsigned(Addr));
    Dout <= memory(temp);

END dataflow;
```

Describing Combinational Logic Using Dataflow Design Style

MLU Example

MLU Block Diagram



MLU: Entity Declaration

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mlu IS
    PORT(
        NEG_A : IN STD_LOGIC;
        NEG_B : IN STD_LOGIC;
        NEG_Y : IN STD_LOGIC;
        A :     IN STD_LOGIC;
        B :     IN STD_LOGIC;
        L1 :    IN STD_LOGIC;
        L0 :    IN STD_LOGIC;
        Y :     OUT STD_LOGIC
    );
END mlu;
```

MLU: Architecture Declarative Section

```
ARCHITECTURE  mlu_dataflow OF mlu IS

SIGNAL  A1 :  STD_LOGIC;
SIGNAL  B1 :  STD_LOGIC;
SIGNAL  Y1 : STD_LOGIC;
SIGNAL  MUX_0 : STD_LOGIC;
SIGNAL  MUX_1 : STD_LOGIC;
SIGNAL  MUX_2 : STD_LOGIC;
SIGNAL  MUX_3 : STD_LOGIC;
SIGNAL  L: STD_LOGIC_VECTOR(1 DOWNTO 0);
```

MLU - Architecture Body

```
BEGIN
    A1<= NOT A  WHEN (NEG_A='1') ELSE
         A;
    B1<= NOT B  WHEN (NEG_B='1') ELSE
         B;
    Y <= NOT Y1 WHEN (NEG_Y='1') ELSE
         Y1;

    MUX_0 <= A1  AND  B1;
    MUX_1 <= A1   OR   B1;
    MUX_2 <= A1  XOR  B1;
    MUX_3 <= A1  XNOR B1;

    L <= L1 & L0;

    with (L) select
       Y1 <=  MUX_0  WHEN "00",
              MUX_1  WHEN "01",
              MUX_2  WHEN  "10",
              MUX_3  WHEN OTHERS;

END mlu_dataflow;
```

Logic Implied Most Often by Conditional and Selected Concurrent Signal Assignments

Data-flow VHDL

Major instructions

Concurrent statements

- concurrent signal assignment ( )
- **conditional concurrent signal assignment (when-else)**
- selected concurrent signal assignment (with-select-when)

Conditional concurrent signal assignment

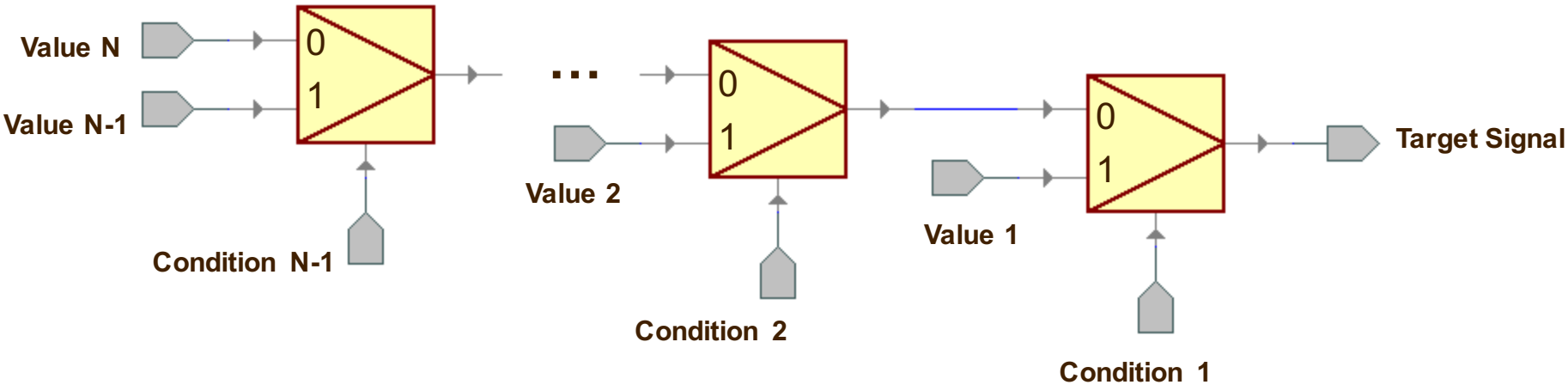
When - Else

```
target_signal <= value1 when condition1 else
                 value2 when condition2 else
                   . . .
                 valueN-1 when conditionN-1 else
                 valueN;
```

Most often implied structure

When - Else

```
target_signal <= value1 when condition1 else
                 value2 when condition2 else
                    . . .
                 valueN-1 when conditionN-1 else
                 valueN;
```



Data-flow VHDL

Major instructions

Concurrent statements

- concurrent signal assignment ( )
- conditional concurrent signal assignment (when-else)
- **selected concurrent signal assignment (with-select-when)**

Selected concurrent signal assignment

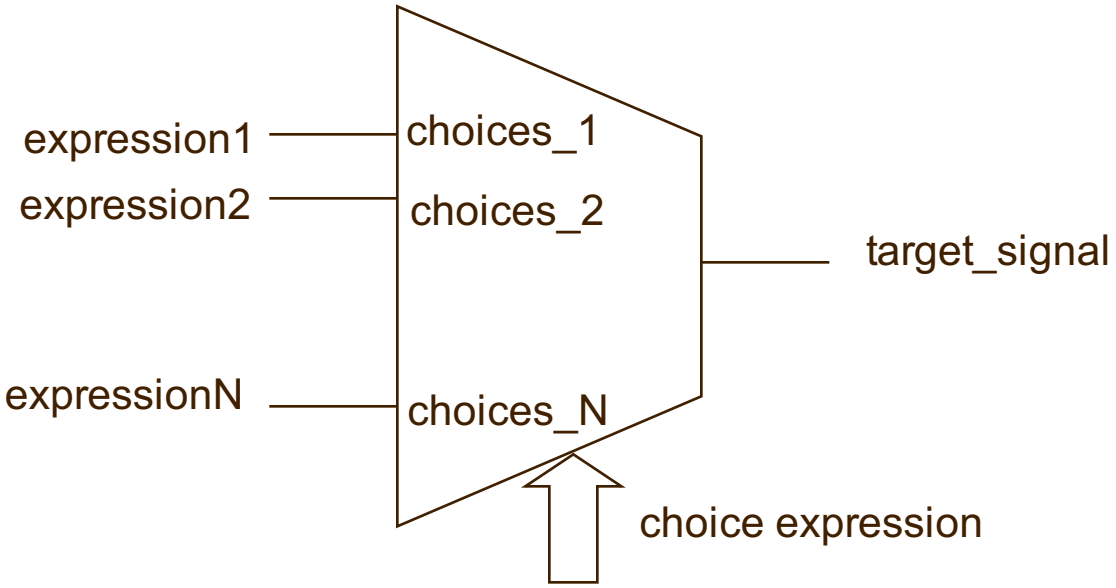
With –Select-When

```
with choice_expression select
   target_signal <= expression1 when choices_1,
                    expression2 when choices_2,
                        . . .
                    expressionN when choices_N;
```

Most Often Implied Structure

With –Select-When

```
with choice_expression select
   target_signal <= expression1 when choices_1,
                    expression2 when choices_2,
                        . . .
                    expressionN when choices_N;
```



Allowed formats of *choices_k*

WHEN value

WHEN value_1 | value_2 | .... | value N

WHEN OTHERS

Allowed formats of *choice_k - example*

```
WITH sel SELECT
     y <= a WHEN "000",
          c WHEN "001" | "111",
          d WHEN OTHERS;
```

when-else vs. with-select-when (1)

"when-else" should be used when:

1) there is only one condition (and thus, only one else), as in the 2-to-1 MUX

2) conditions are independent of each other (e.g., they test values of different signals)

3) conditions reflect priority (as in priority encoder); one with the highest priority need to be tested first.

when-else vs. with-select-when (2)

"with-select-when" should be used when there is

1) more than one condition
2) conditions are closely related to each other (e.g., represent different ranges of values of the same signal)
3) all conditions have the same priority (as in the 4-to-1 MUX).